Concurrency

Edward Z. Yang

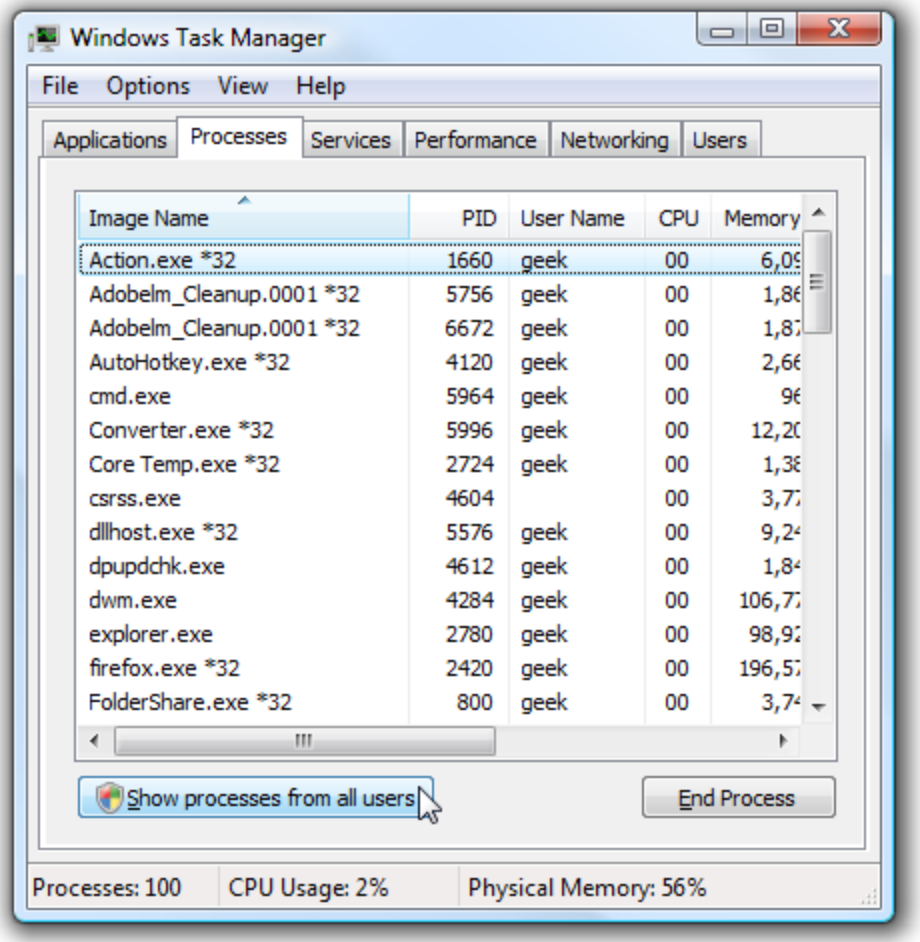
What is concurrency?

"It's when you have multiple threads"

"Things execute at the same time"

Why concurrency?

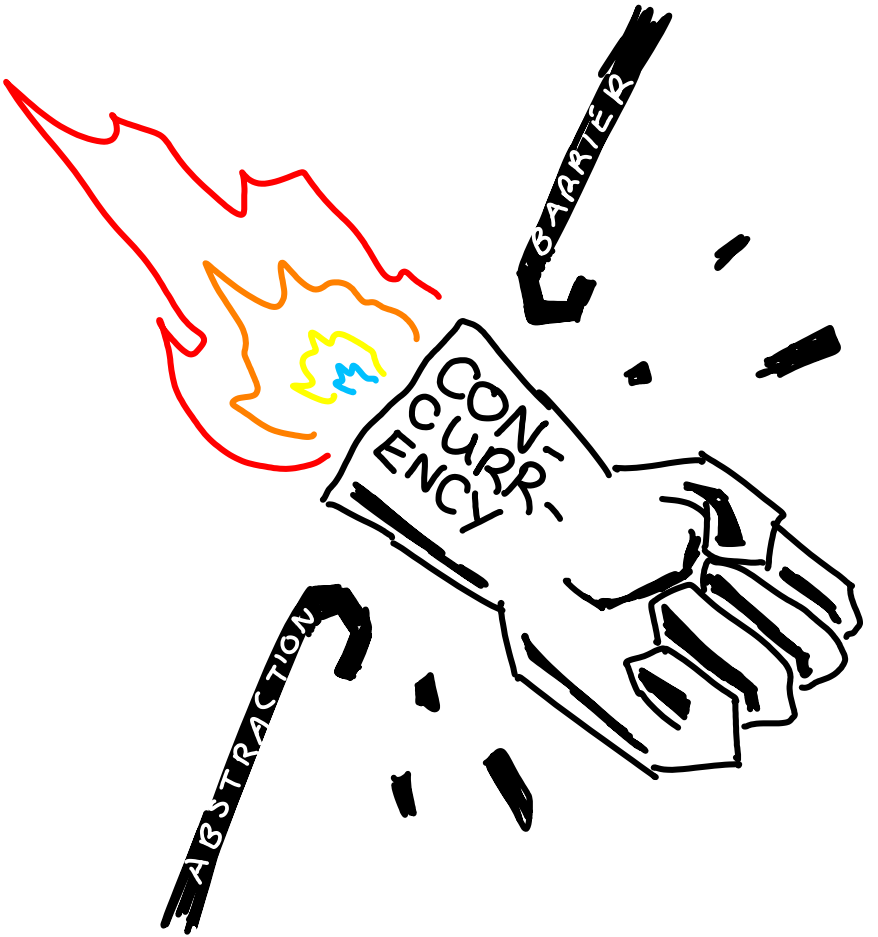
Concurrency is ubiquitous

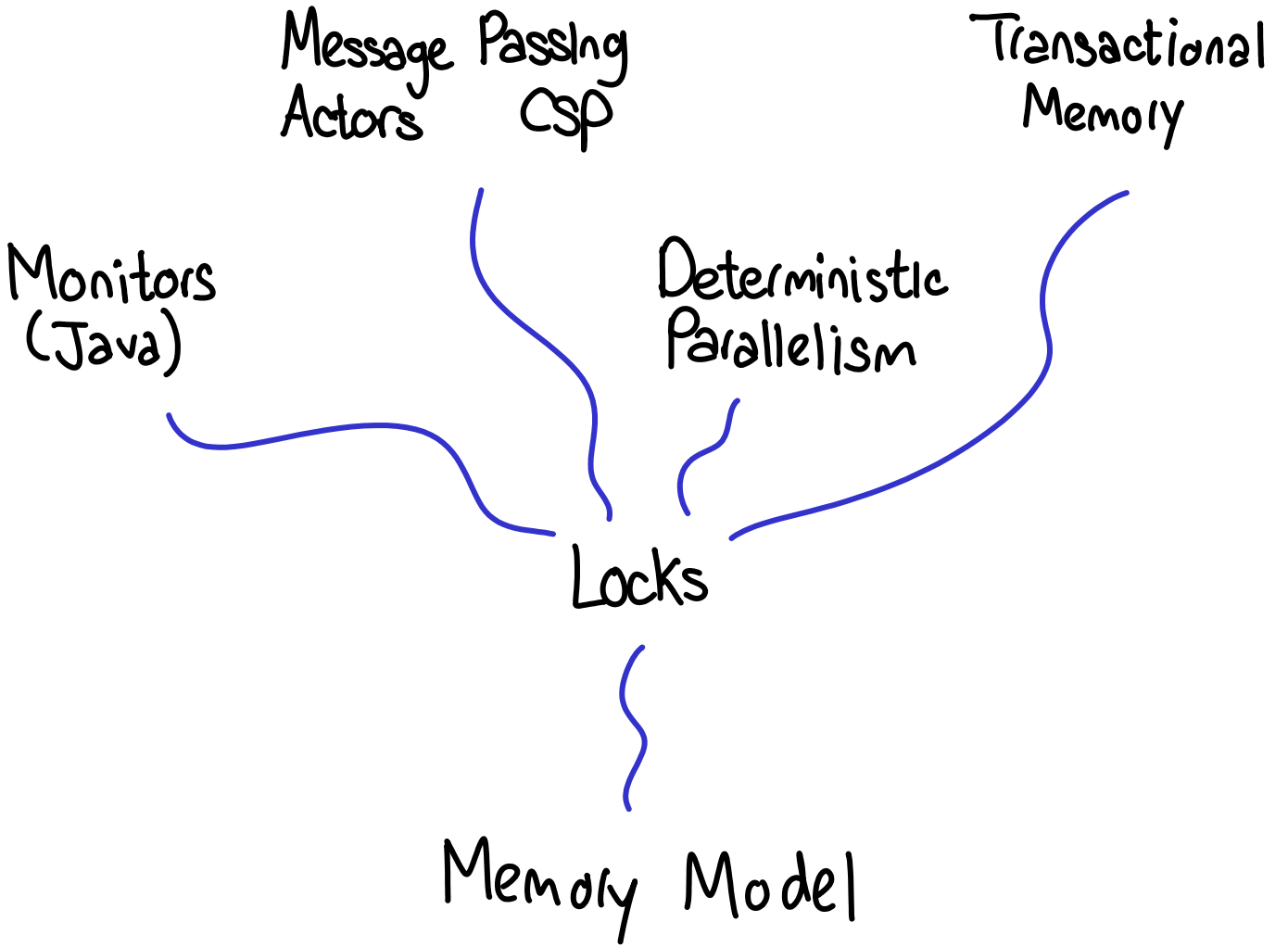


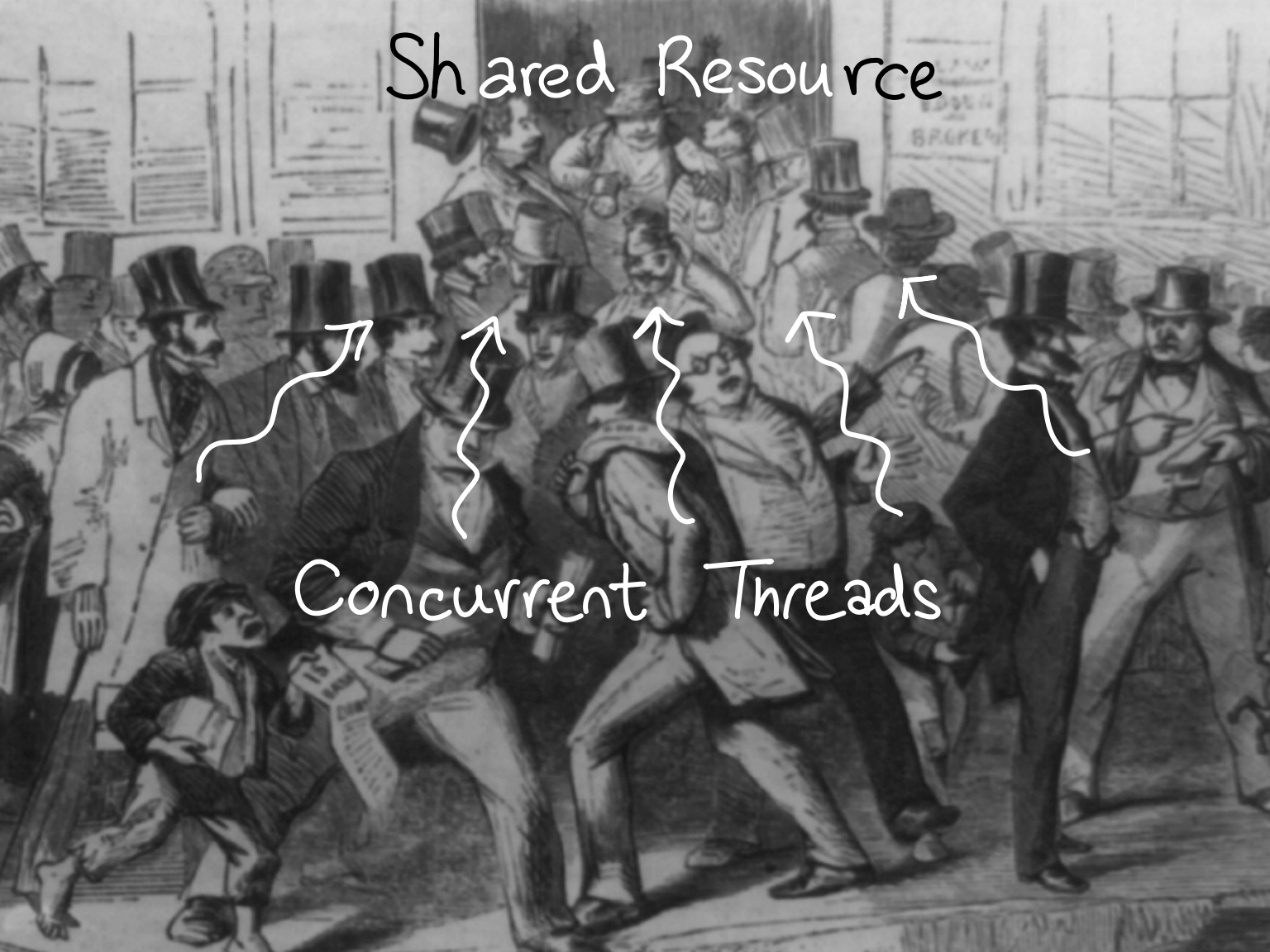
Concurrency is ubiquitous

How can programming languages make concurrent programming easier?

What abstractions are most effective?


BARRIER
CON-
CURR-
ENCY
ABSTRACTION


Message Passing
Actors
CSP
Transactional
Memory
Monitors
(Java)
Deterministic
Parallelism
Locks
Memory Model


Shared Resource
Concurrent Threads

Memory = Shared Resource

balance := 100

Thread #1

```
if (balance ≥ 100) {
    balance -= 100;
} else {
    insufficient funds
}
```

Thread #2

```
if (balance ≥ 50) {
    balance -= 50;
} else {
    insufficient funds
}
```

Memory = Shared Resource

balance := 100

Thread #1

```
if (balance ≥ 100) {
    balance -= 100;
} else {
    insufficient funds
}
```

a "critical section"
... at any moment, only one of these... processes is in its critical section

Dijkstra 1965


Shared Resource
Waiting Threads
Thread in Critical Region

a "critical section"
... at any moment, only one of these... processes is in its critical section

Dijkstra 1965

I will tell you how to implement a lock.

But Dijkstra, how do I implement this?

impressively, Dijkstra did this without assuming any sort of compare-and-swap instruction

We have locks, can we go home?

(or atomic test-and-set, or semaphores)

Where do I put locks?

And what should happen if I leave out locks?

Thread #1	Thread #2
lock(1)	tmp := x+y
x := x+1	
y := y+1	
unlock(1)	

Locks should be associated with the resource

How should I schedule requests?

Thread #1	Thread #2
lock(1)	lock(1)
// high priority request	//only run me if buffer
	// is empty
unlock(1)	unlock(1)

Schedule should be user-programmable

What about composability?

Thread #1	Thread #2
lock(1)	lock(2)
lock(2)	lock(1)
// critical region	// critical region
unlock(2)	unlock(1)
unlock(1)	unlock(2)

Must know which locks are taken out and in what order

Deadlock!

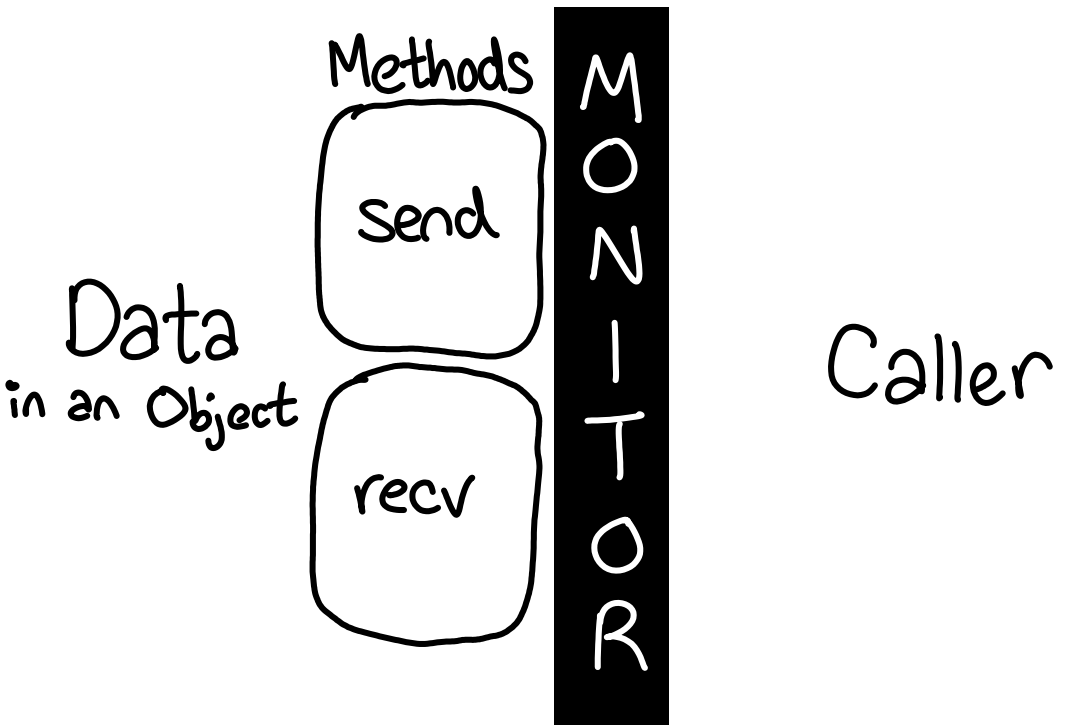
How can programming languages make concurrent programming easier?

By abstractions, of course!

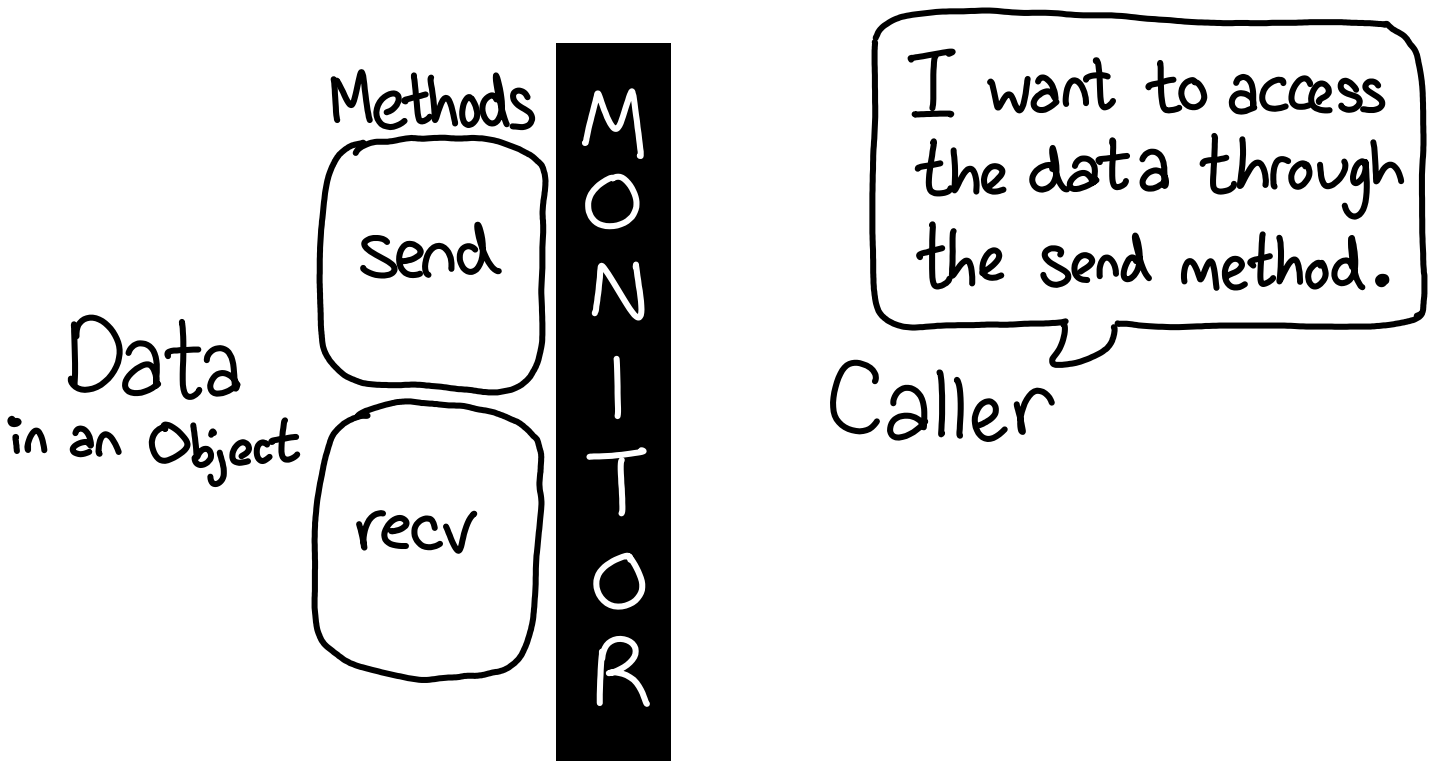
Monitors (Concurrent Pascal, Java)

```
public synchronized T
recv() throws InterruptedException {
    while (queue.isEmpty()) {
        wait();
    }
    return queue.pop();
}

public synchronized void
send(T x) { queue.push(x); notify(); }
```

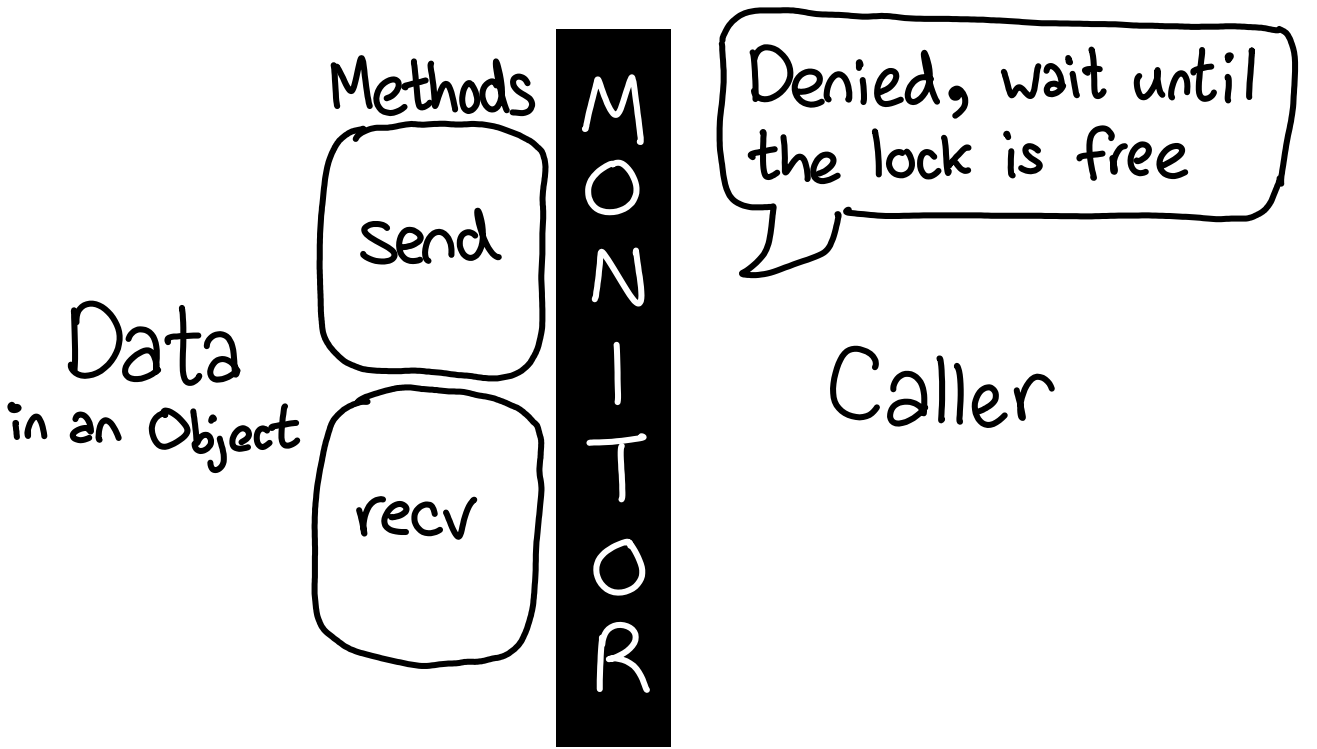
Monitors (Concurrent Pascal, Java)



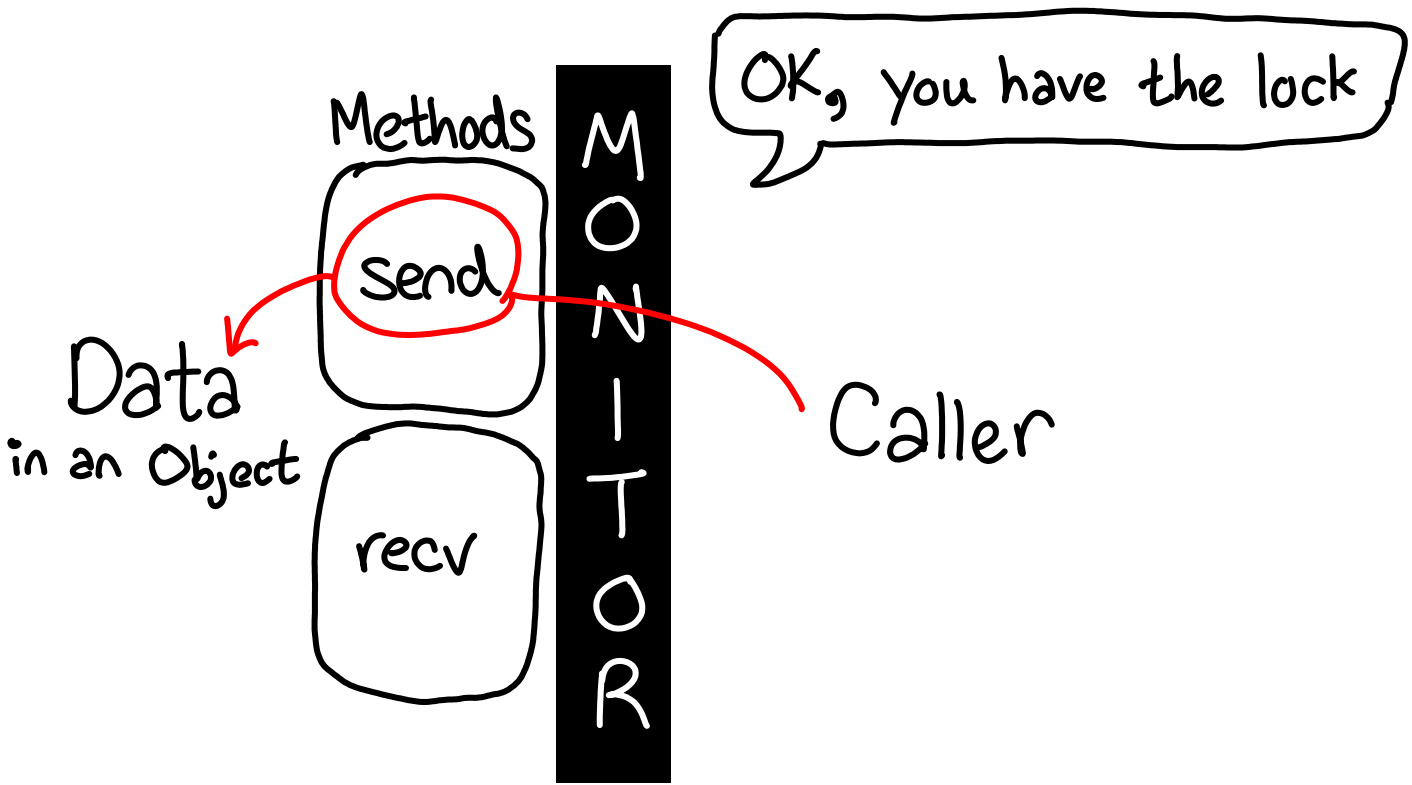
Monitors (Concurrent Pascal, Java)



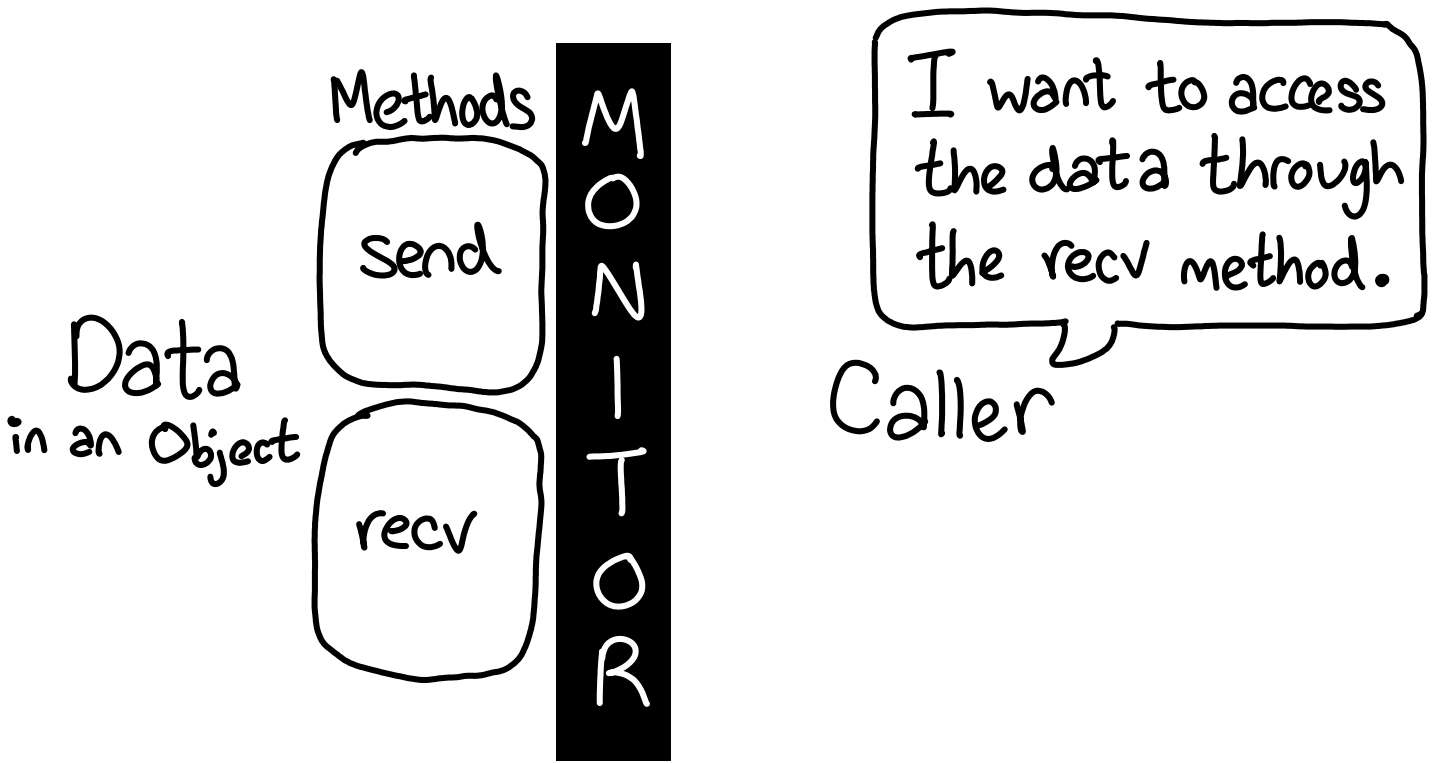
Monitors (Concurrent Pascal, Java)



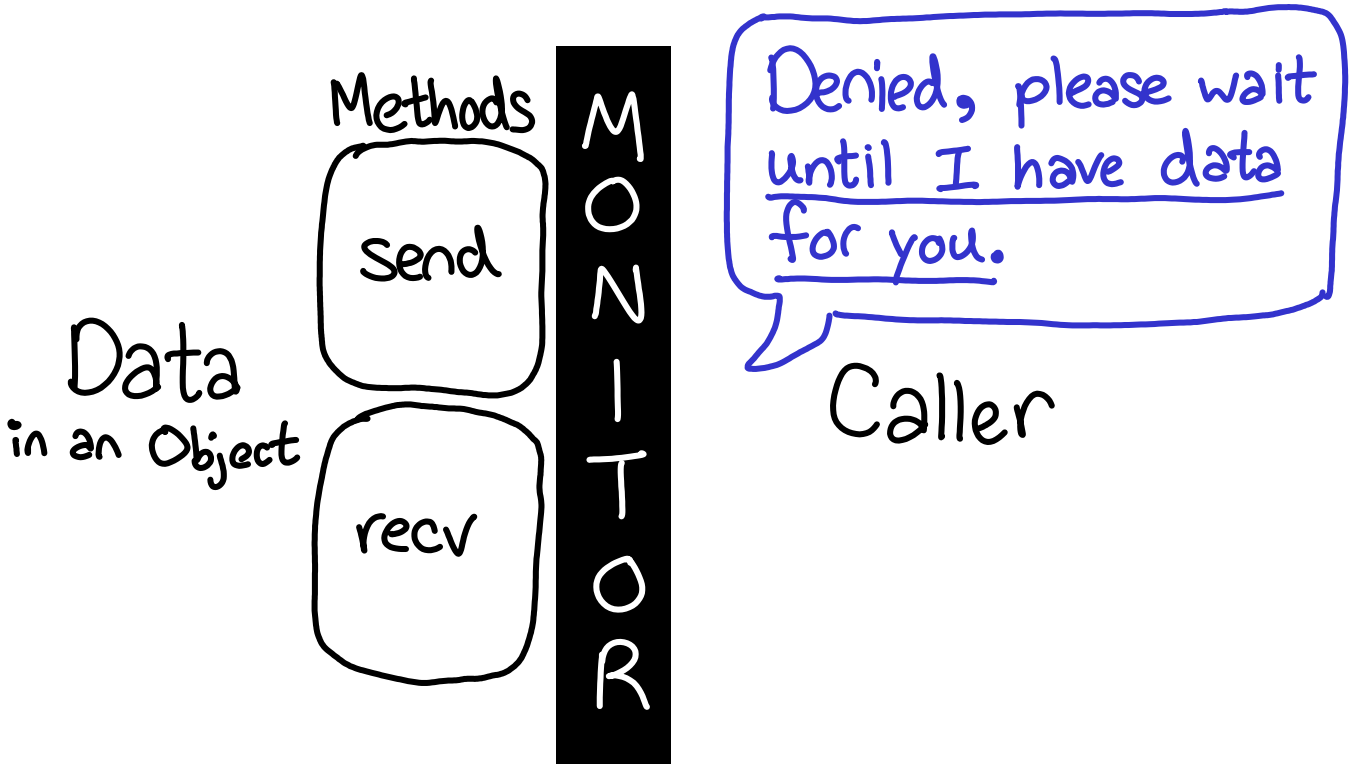
Monitors (Concurrent Pascal, Java)



Monitors (Concurrent Pascal, Java)

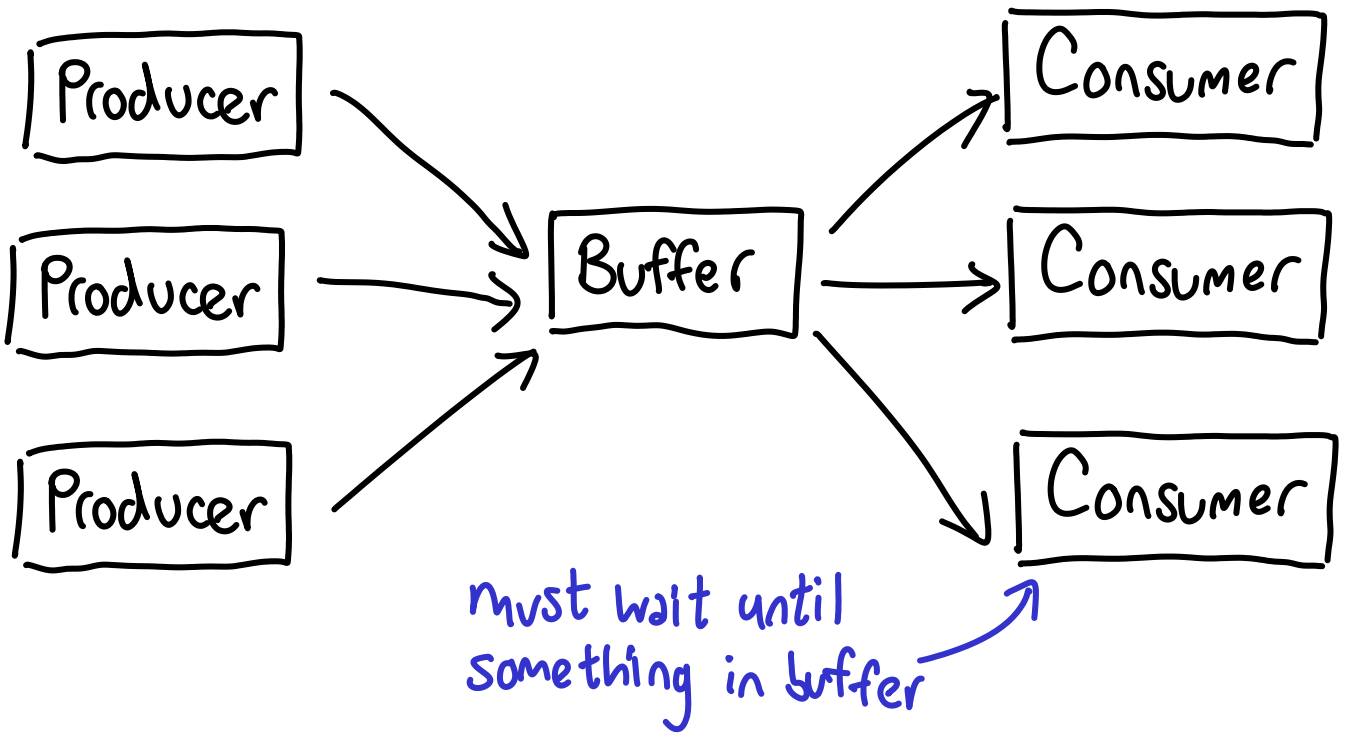


Monitors (Concurrent Pascal, Java)

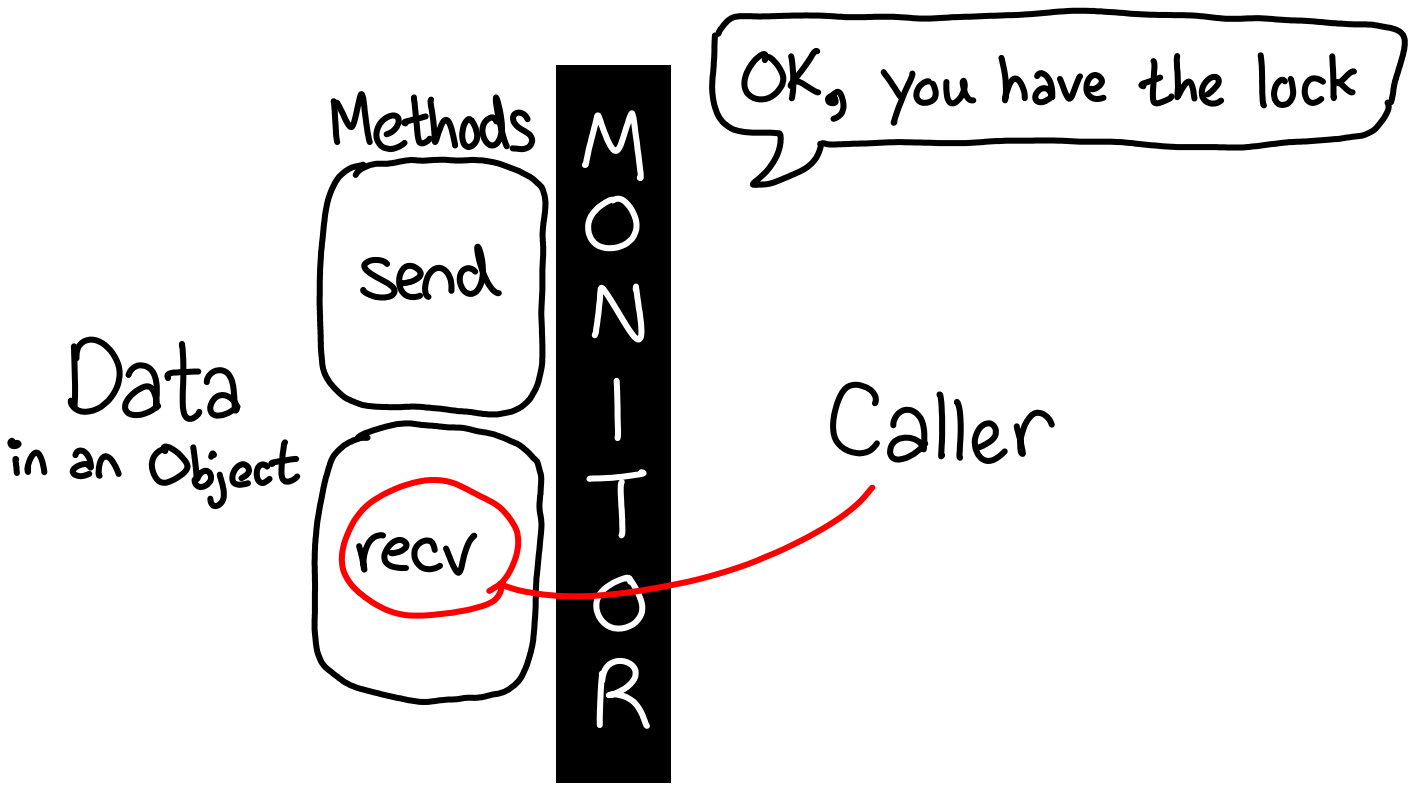


Monitors (Concurrent Pascal, Java)

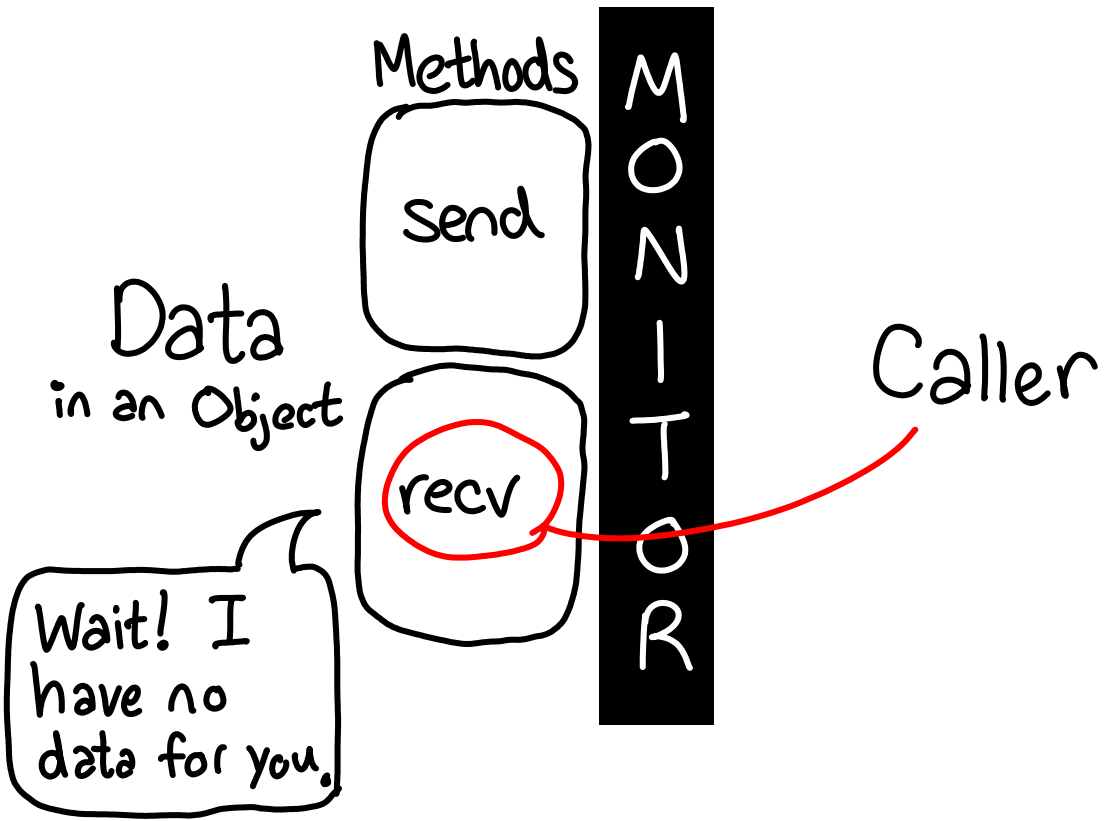
Producer-consumer



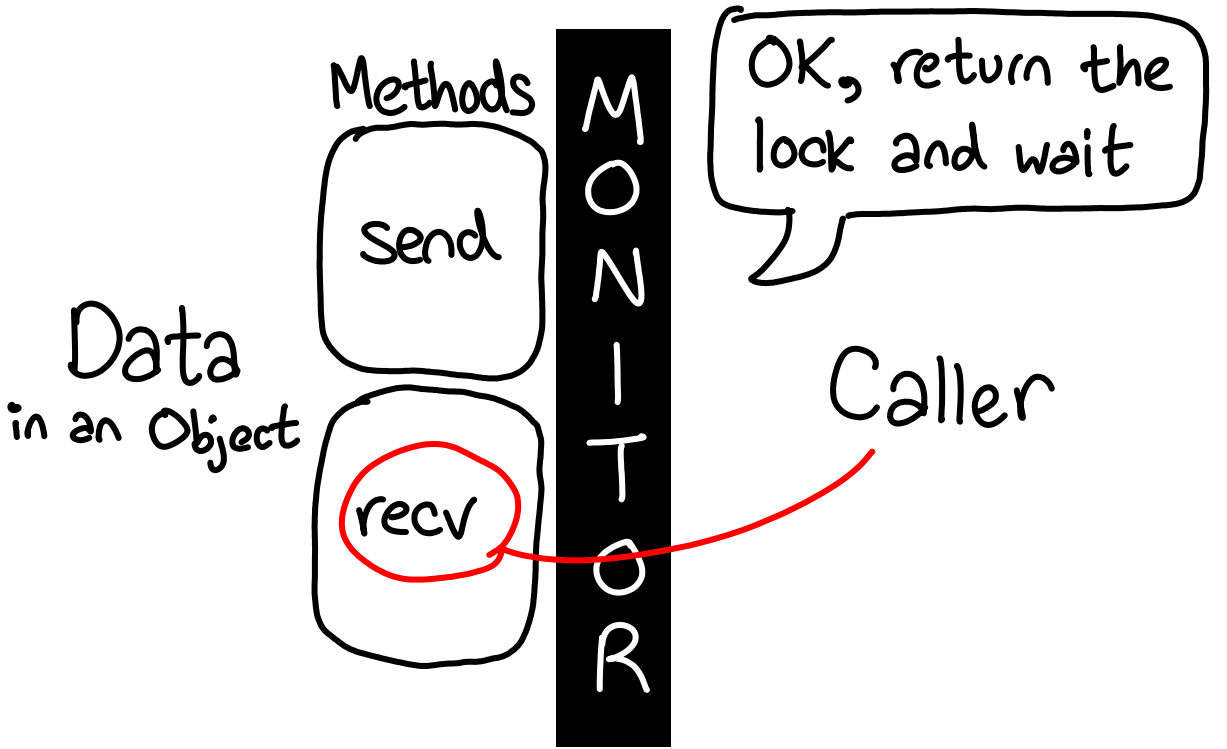
Monitors (Concurrent Pascal, Java)



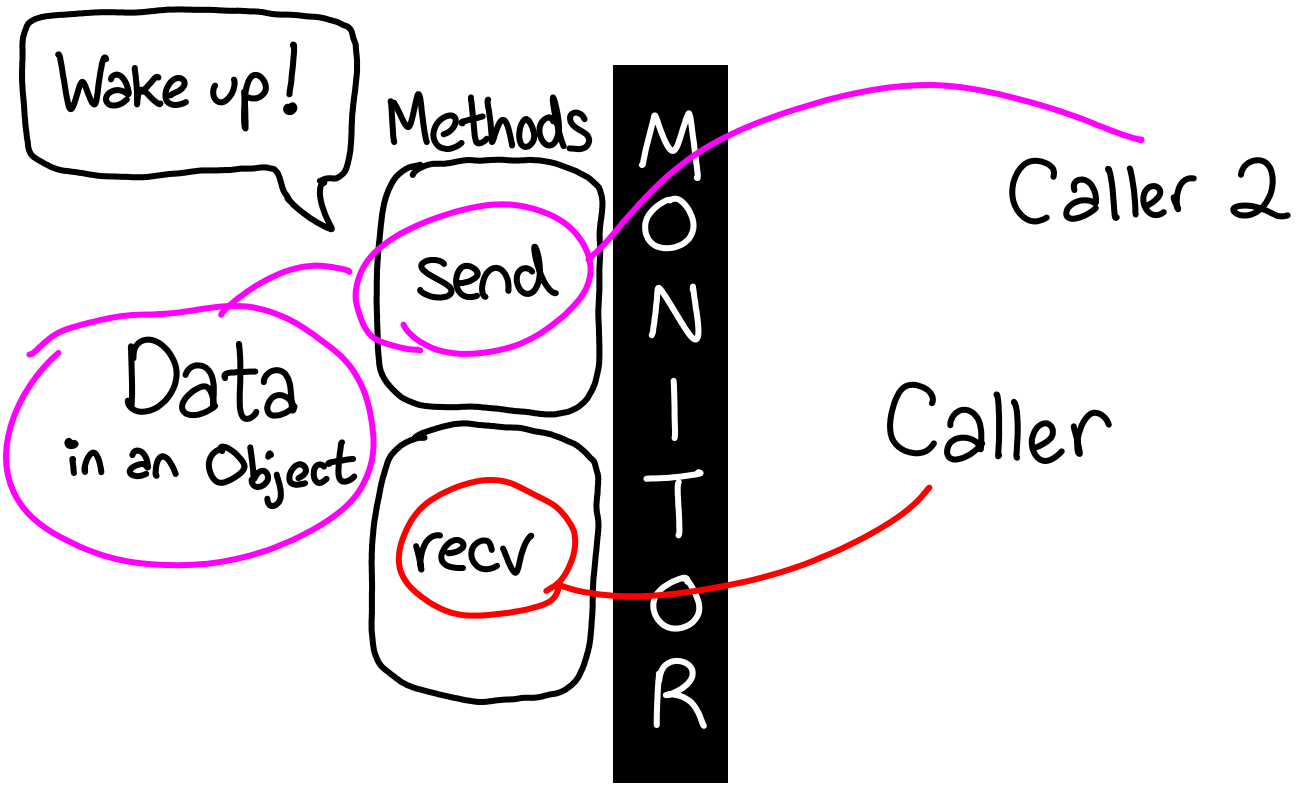
Monitors (Concurrent Pascal, Java)



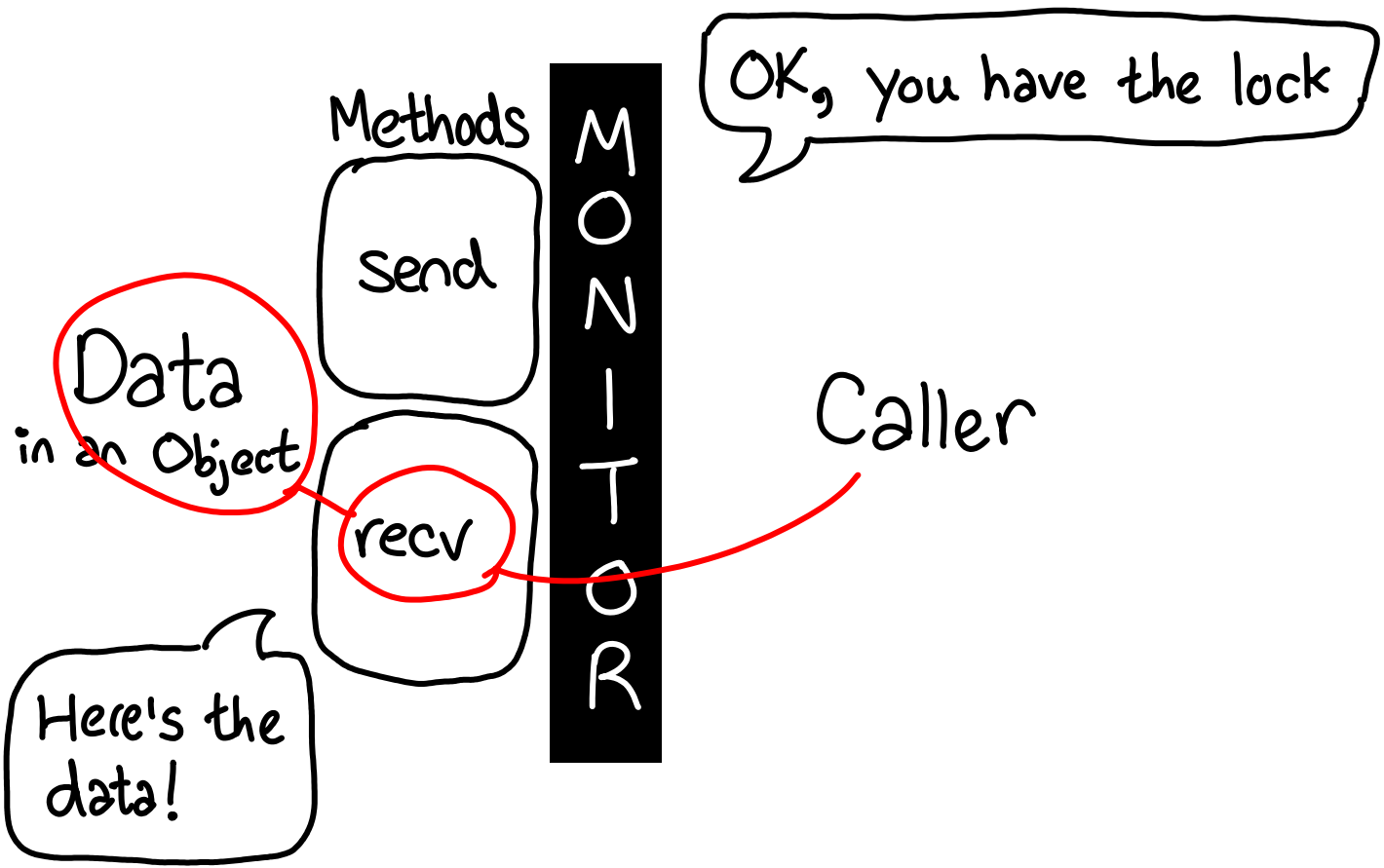
Monitors (Concurrent Pascal, Java)



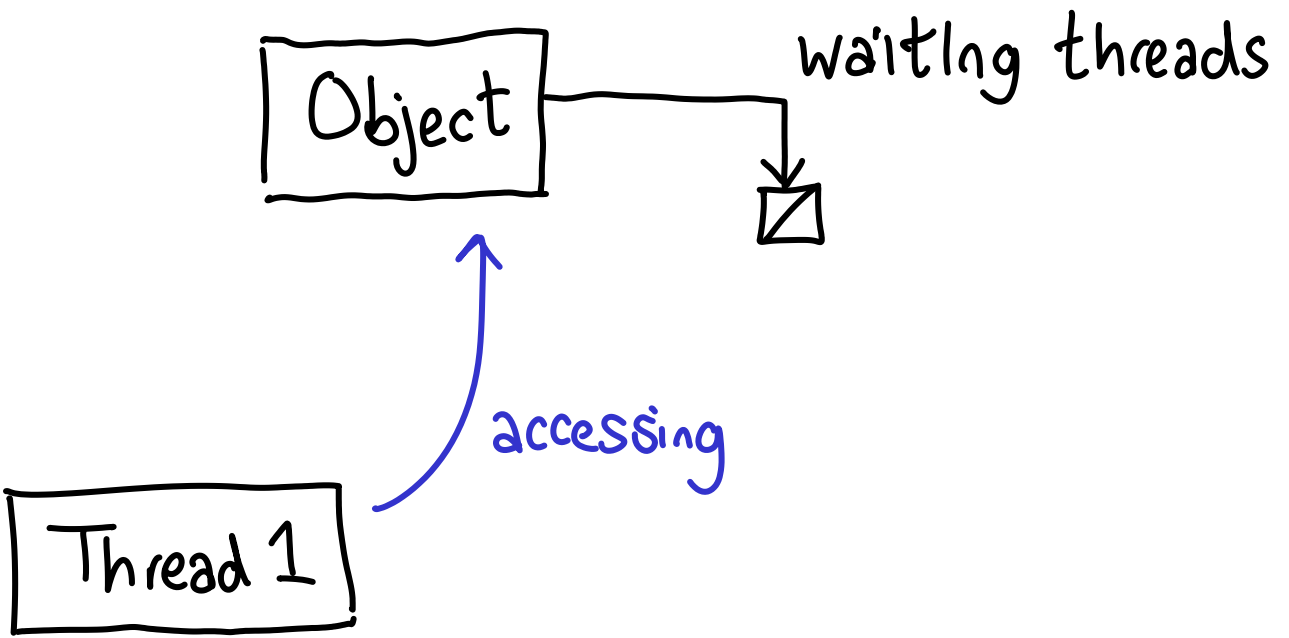
Monitors (Concurrent Pascal, Java)



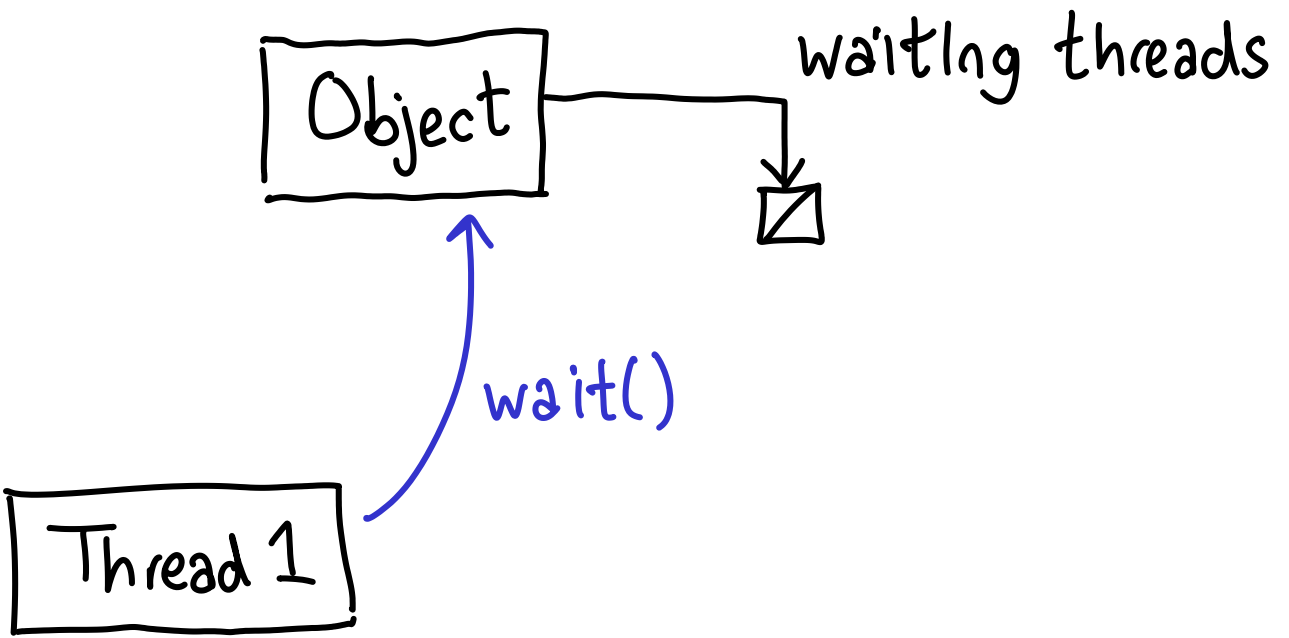
Monitors (Concurrent Pascal, Java)



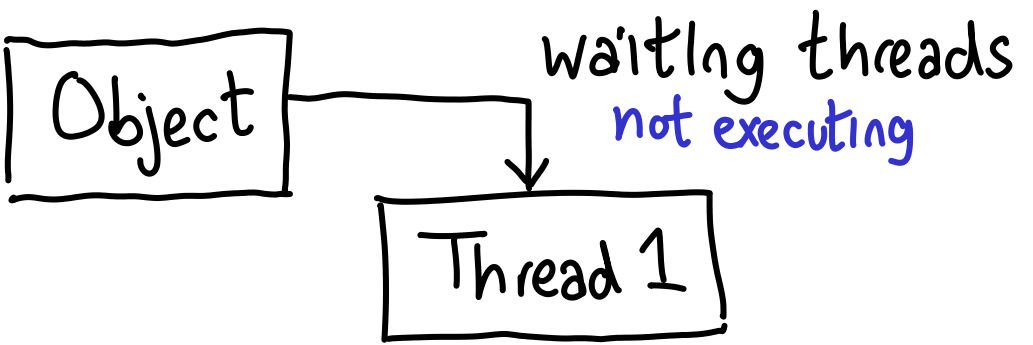
Monitors (Concurrent Pascal, Java)



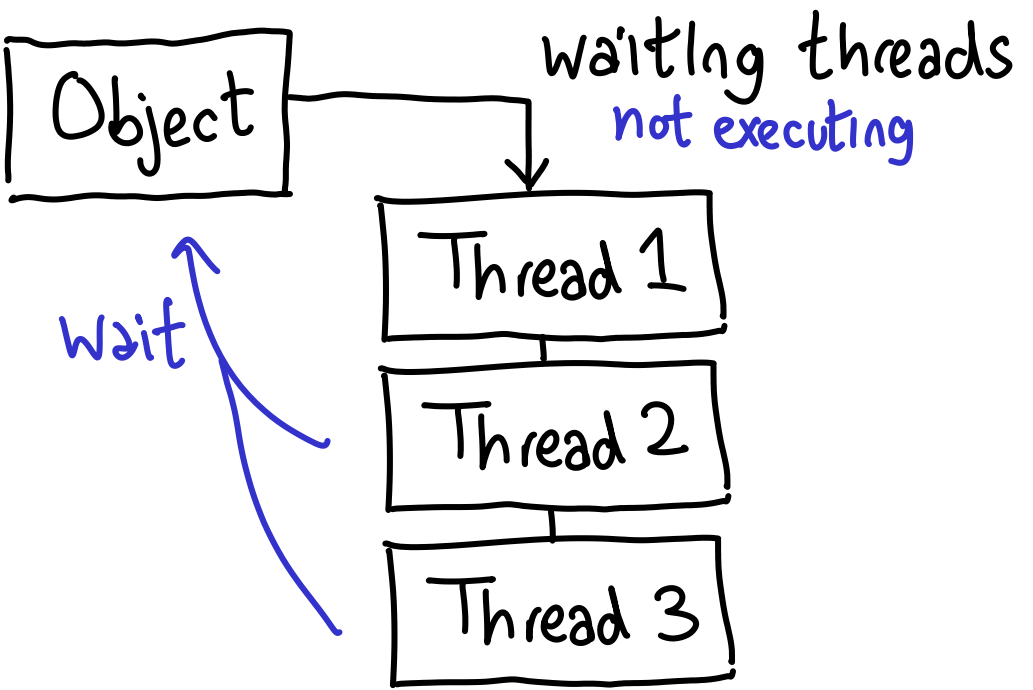
Monitors (Concurrent Pascal, Java)



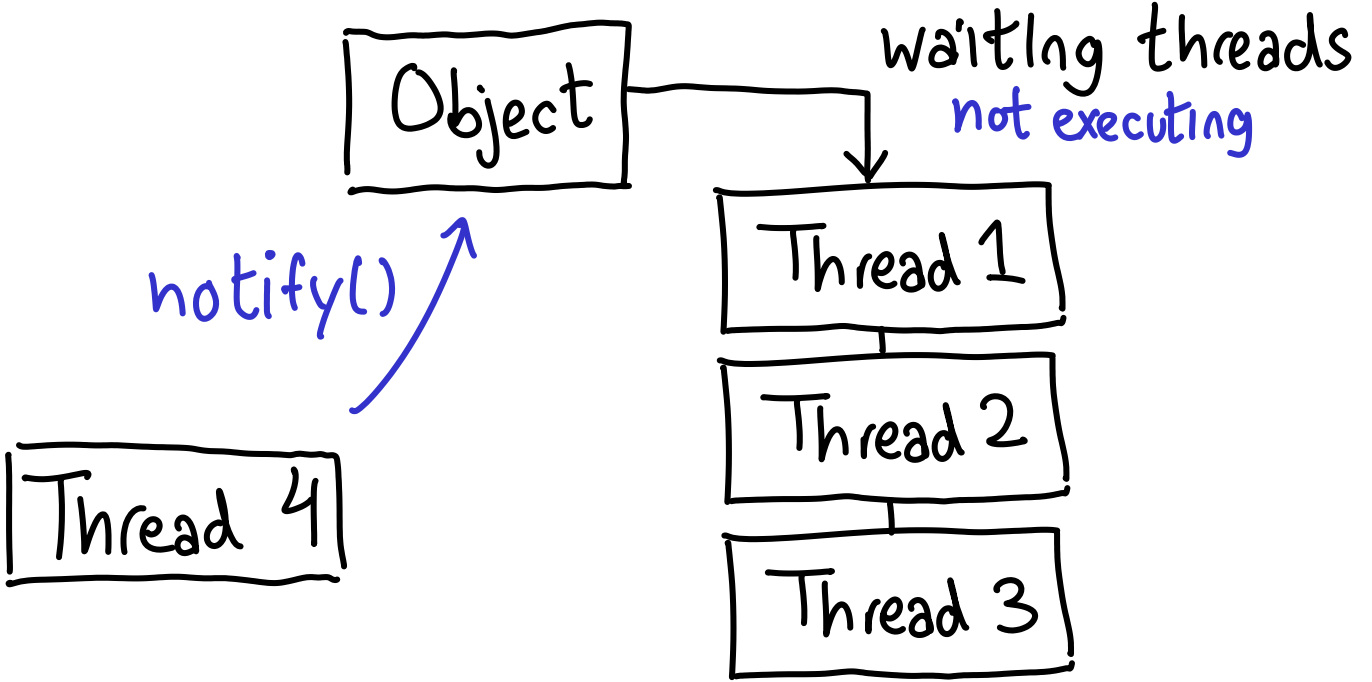
Monitors (Concurrent Pascal, Java)



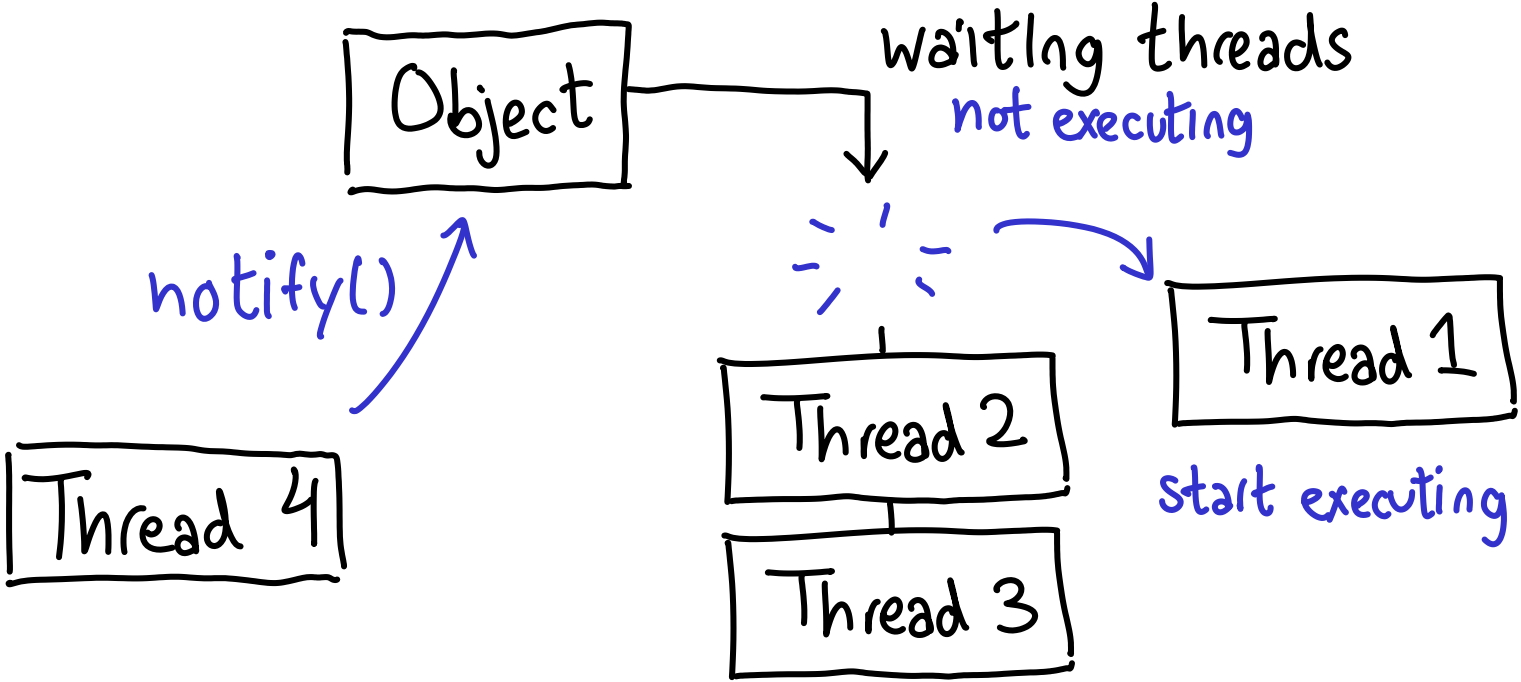
Monitors (Concurrent Pascal, Java)



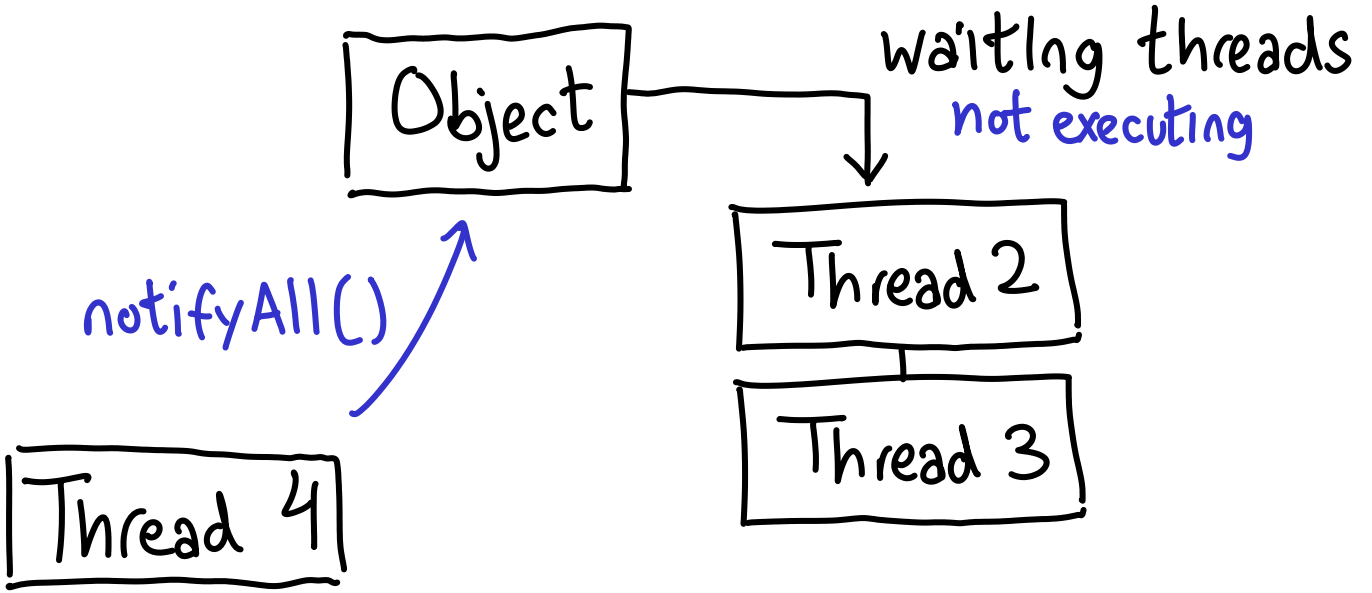
Monitors (Concurrent Pascal, Java)



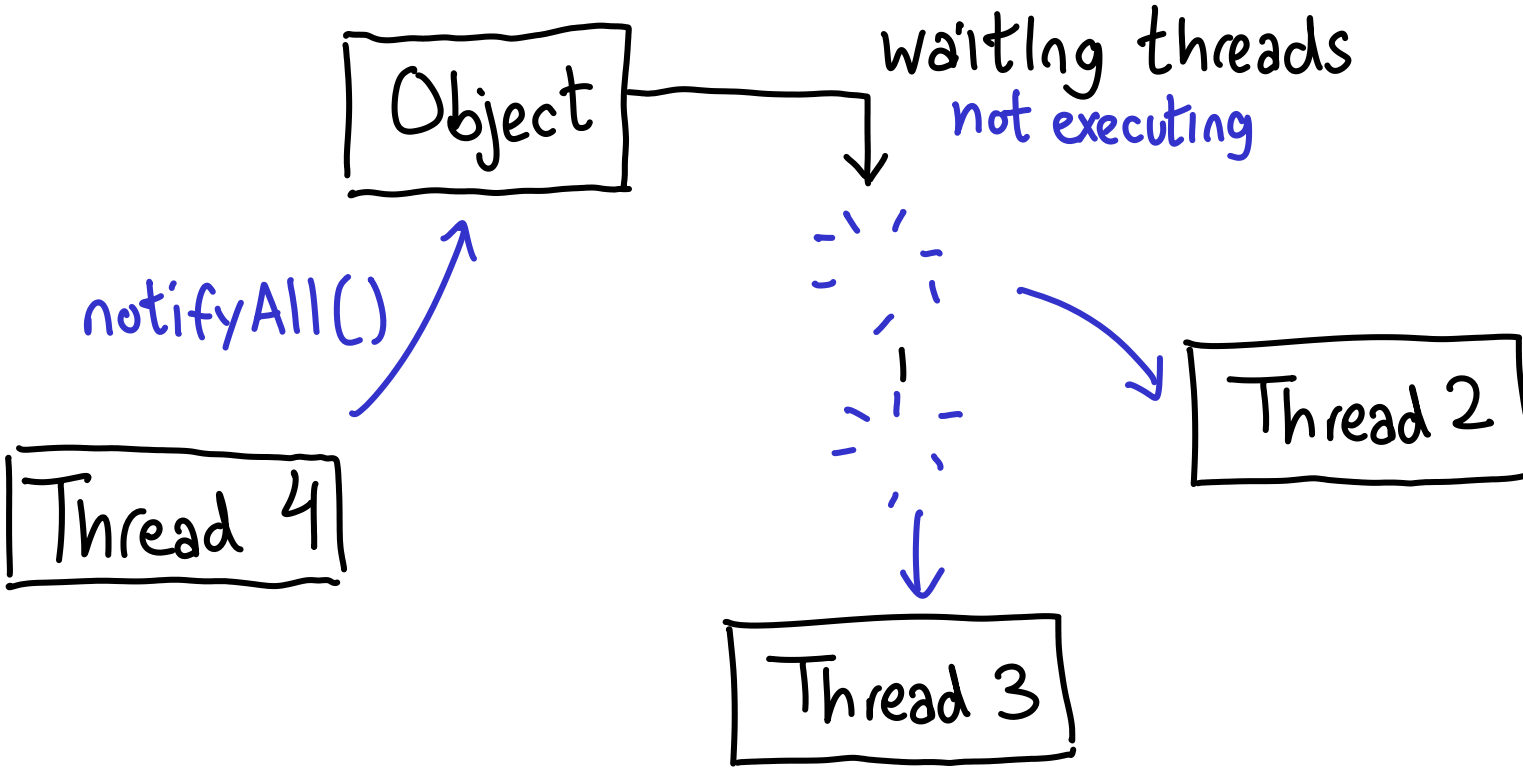
Monitors (Concurrent Pascal, Java)



Monitors (Concurrent Pascal, Java)



Monitors (Concurrent Pascal, Java)



Monitors (Concurrent Pascal, Java)

```
public synchronized T
recv() throws InterruptedException {
    while (queue.isEmpty()) {
        wait();
    }
    return queue.pop();
}

public synchronized void
send(T x) { queue.push(x); notify(); }
```

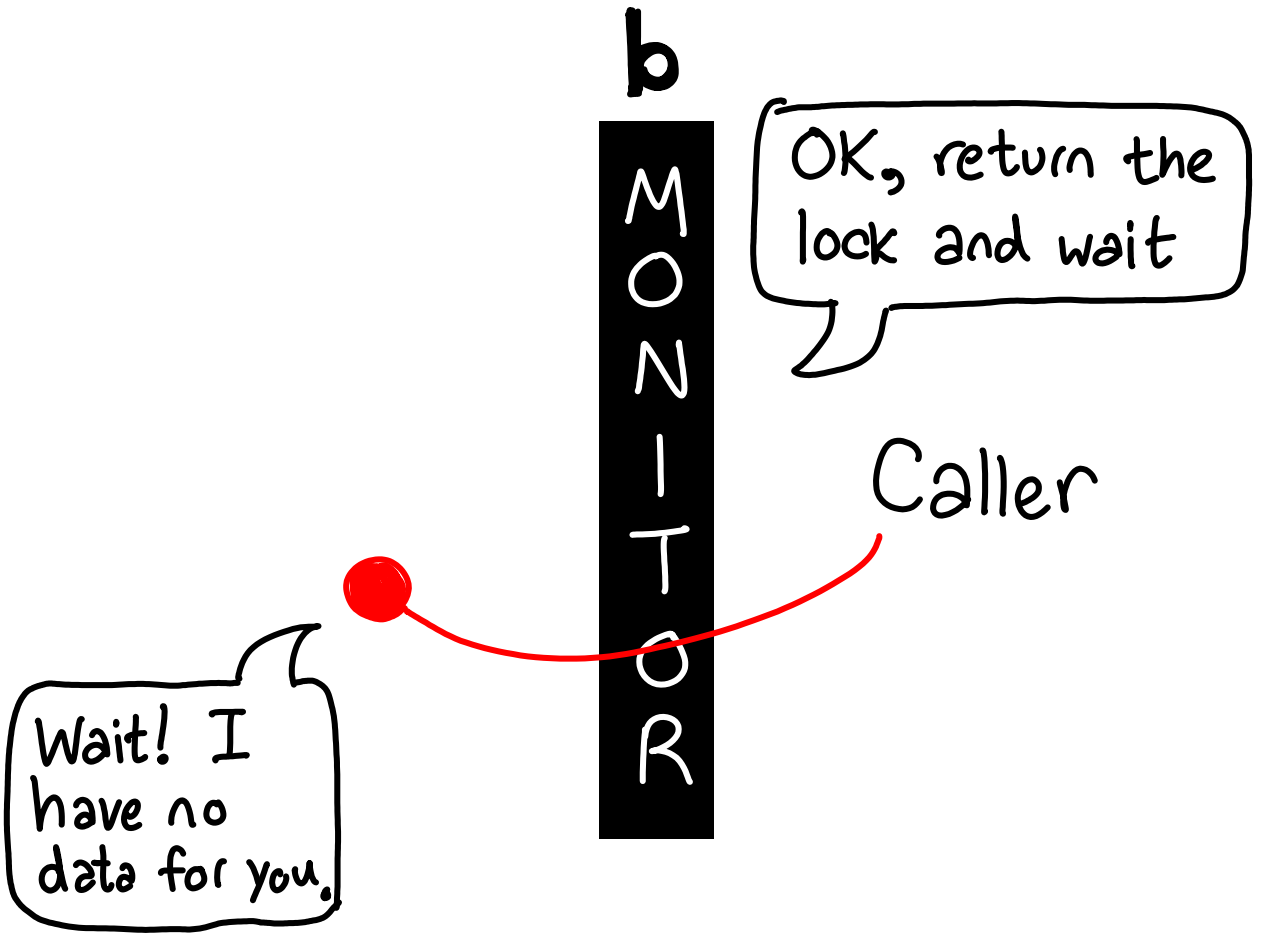
Monitors in Java

- Shared memory concurrency
  programmer responsibility!
- Every object has a lock
- Synchronized / wait / notify / notifyAll
  (Not part of type signature)

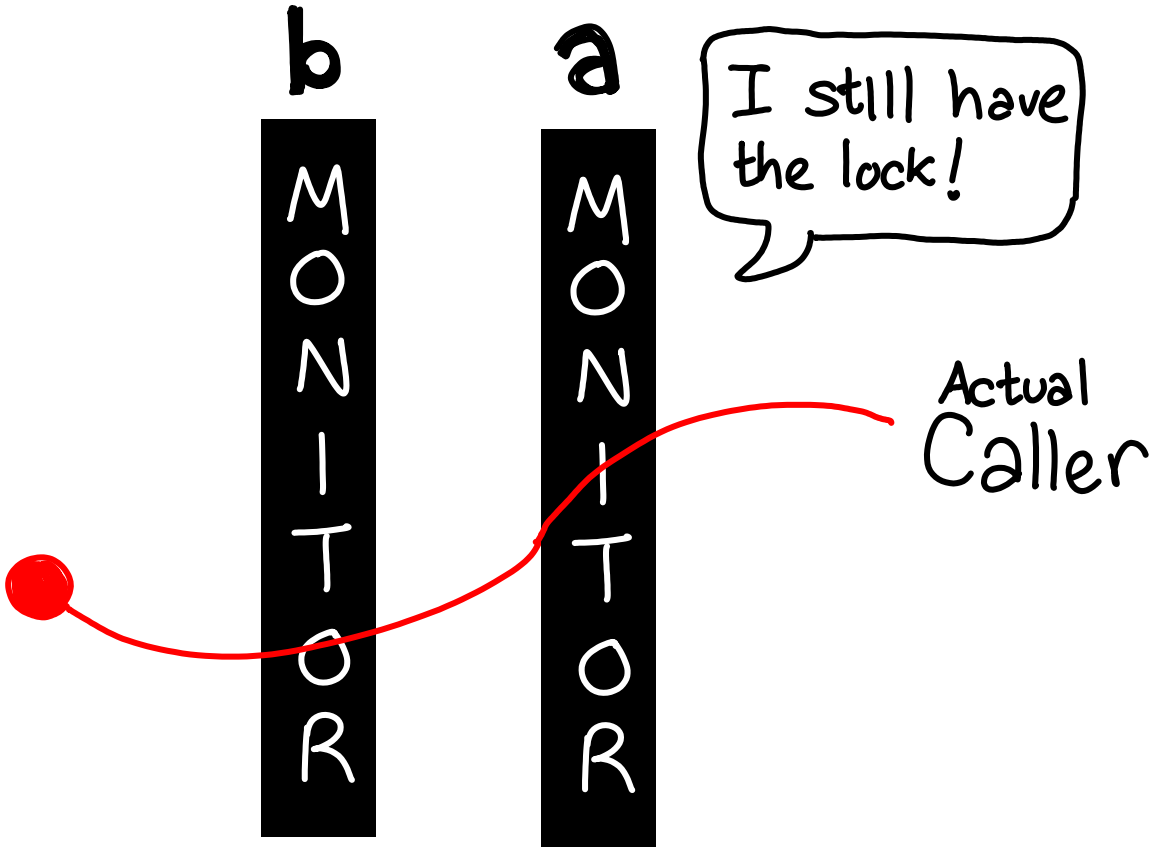
Monitors: Nested Monitor Lockout

```
Synchronized (a) {
    Synchronized (b) {
        a.wait();
    }
}
```

Monitors: Nested Monitor Lockout



Monitors: Nested Monitor Lockout

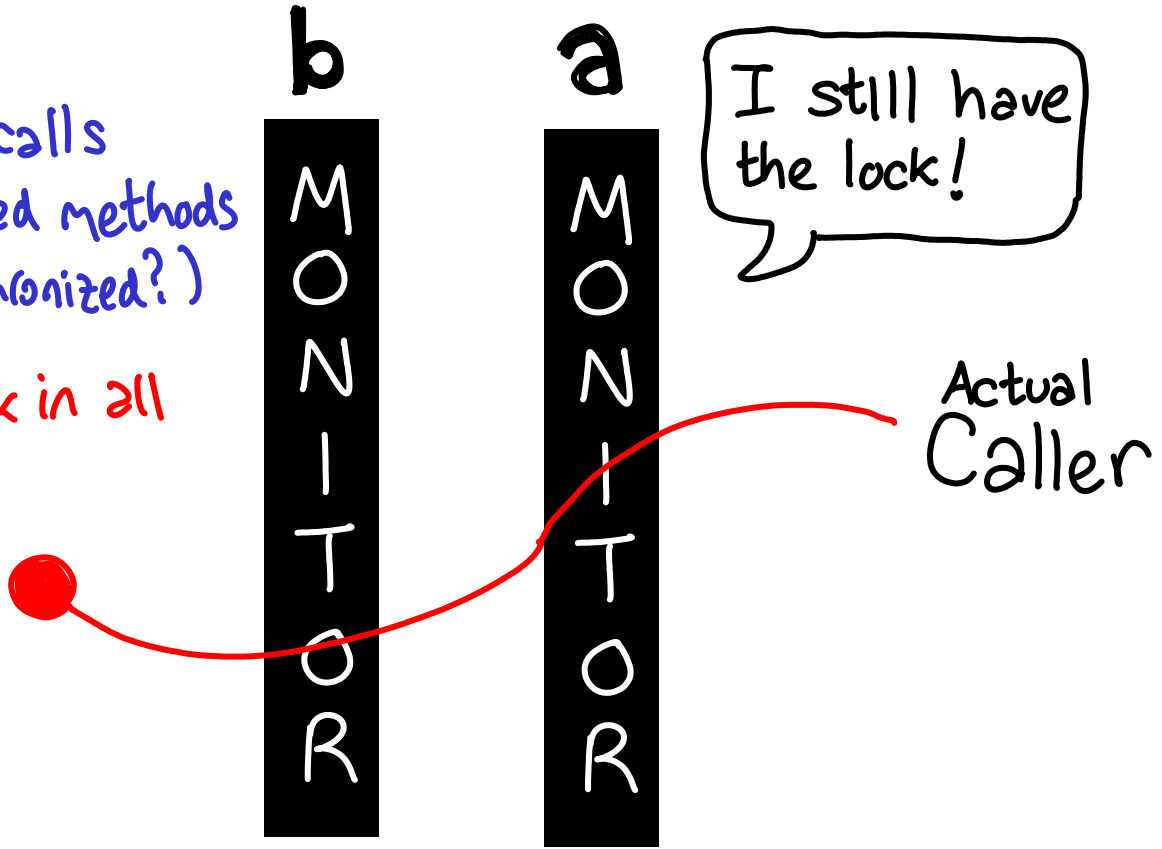


Monitors: Nested Monitor Lockout

"Advice":

No blocking calls in synchronized methods (use un-synchronized?)

Does not work in all situations.

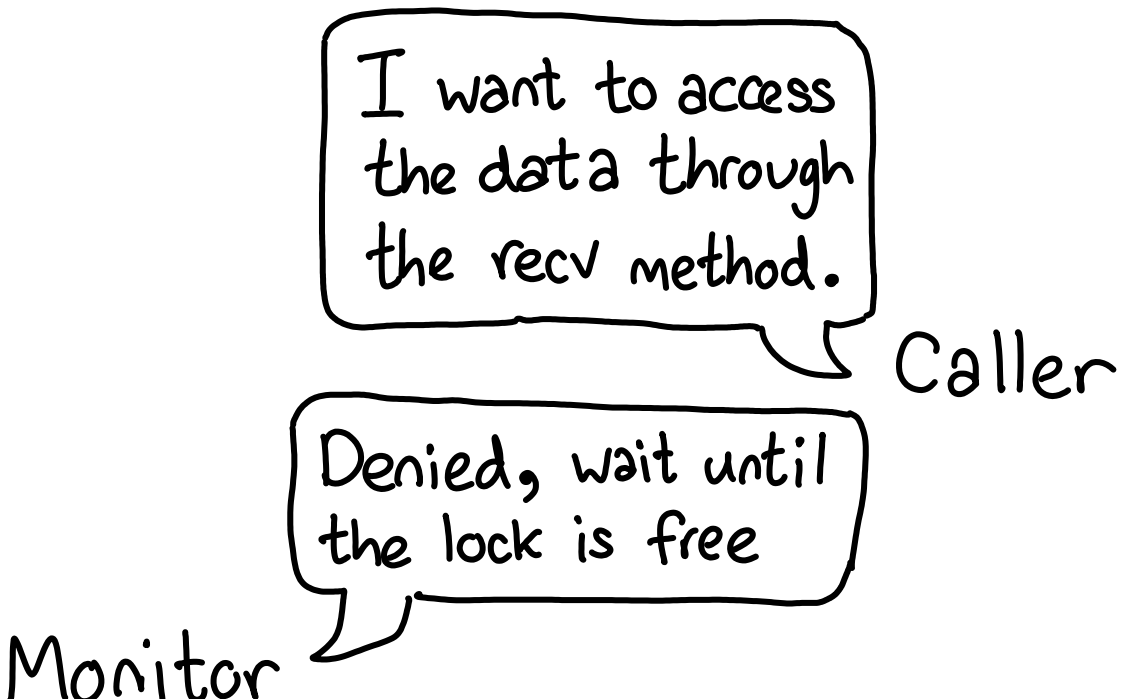


Monitors

synchronized / wait / notify

- ✓ Centralizes and hides synchronization logic associated with data
- ⊗ A bit complicated / a bit inflexible
- ⊗ Nested monitor lockout
- ⊗ Lost notifications
- ⊗ Priority must be supported (Java does not)
- ⚠ Doesn't solve deadlock

Message Passing

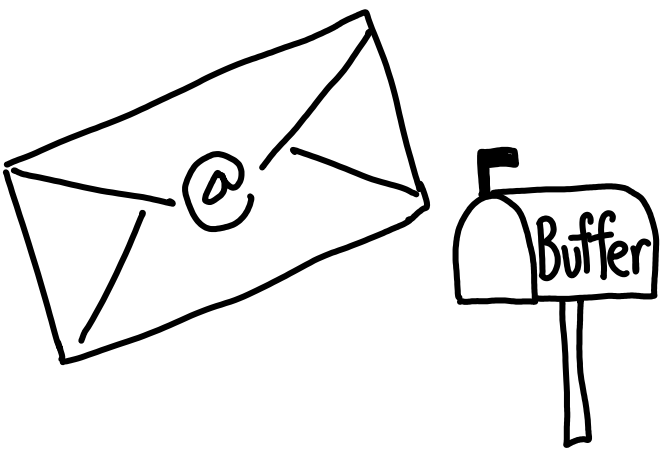


eliminate shared resources

Message Passing

Actor Model

Erlang



Asynchronous

Communicating Sequentlal Processes

Go, Haskell

Synchronous

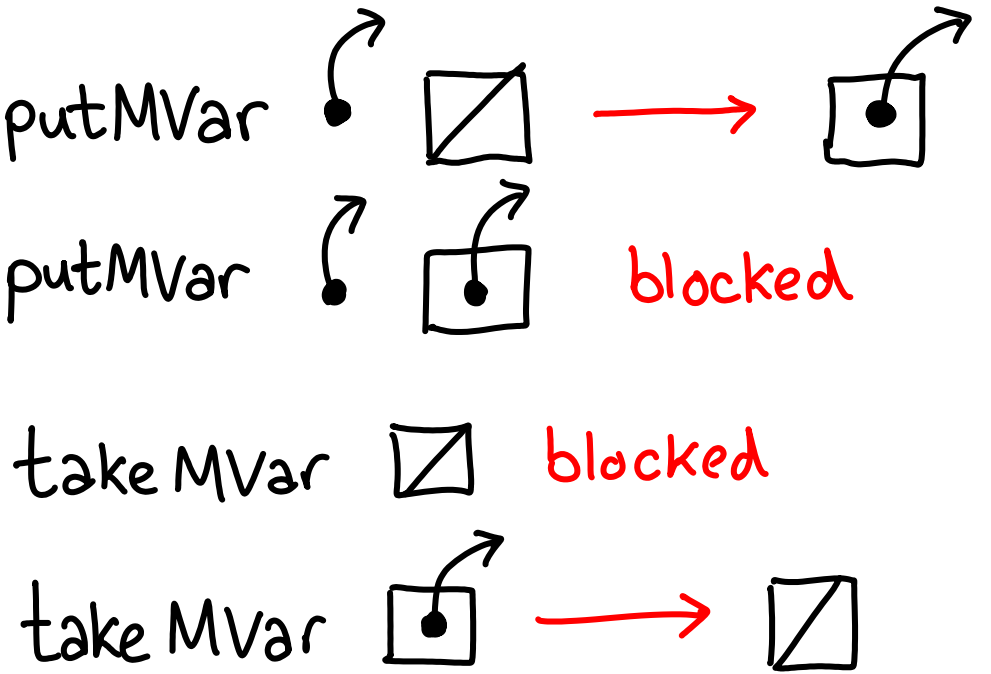
Message Passing: MVars in Haskell

one-place buffers

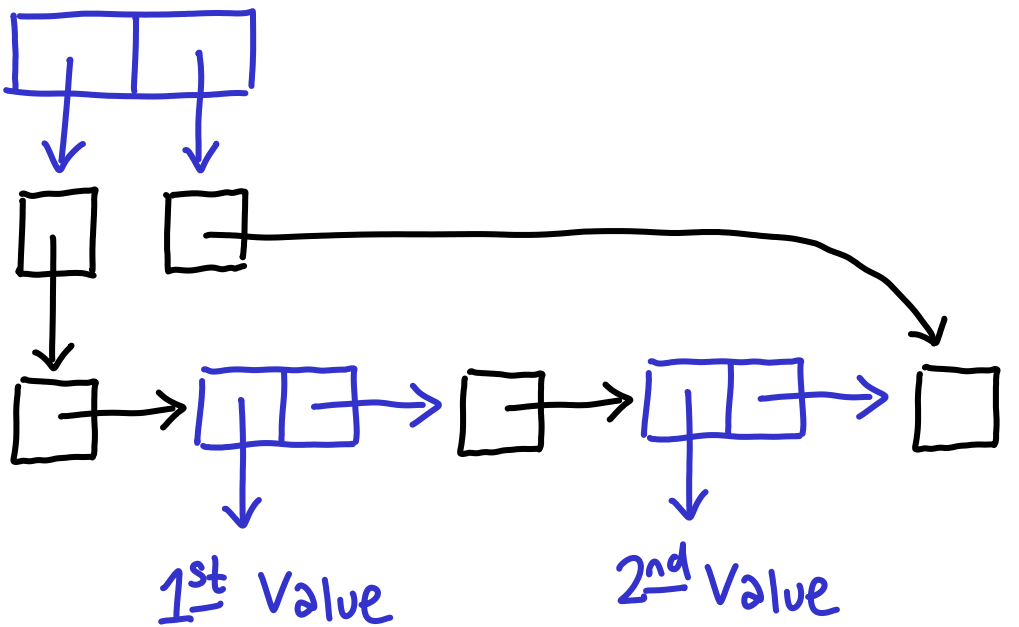
```
data MVar a

newEmptyMVar :: IO (MVar a)
newMVar :: a -> IO (MVar a)
takeMVar :: MVar a -> IO a
putMVar :: MVar a -> a -> IO a
```

Message Passing: MVars in Haskell

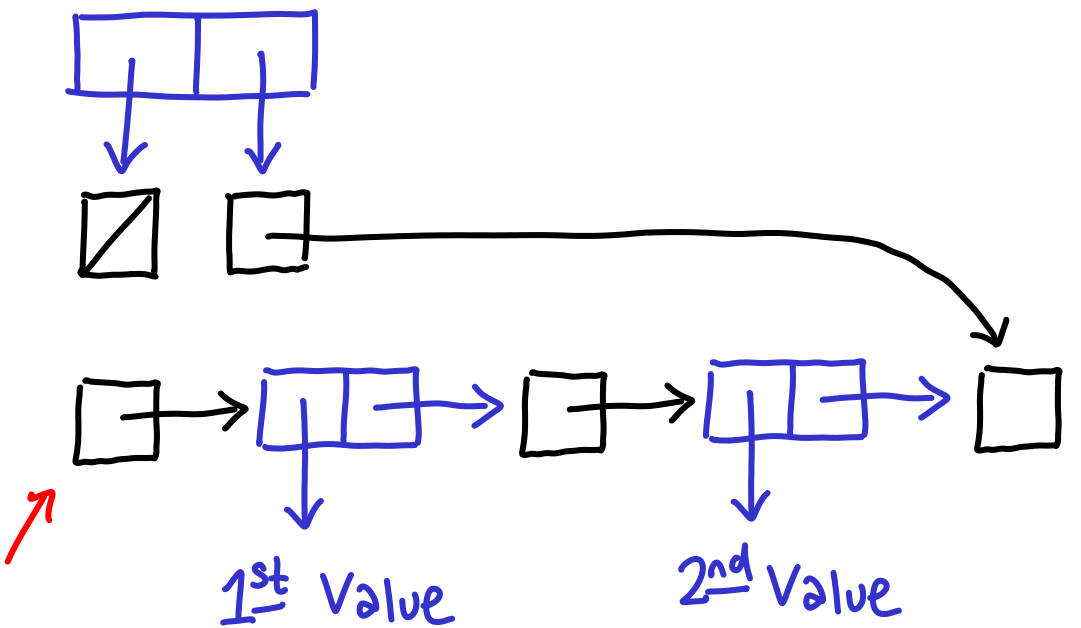
one-place buffers



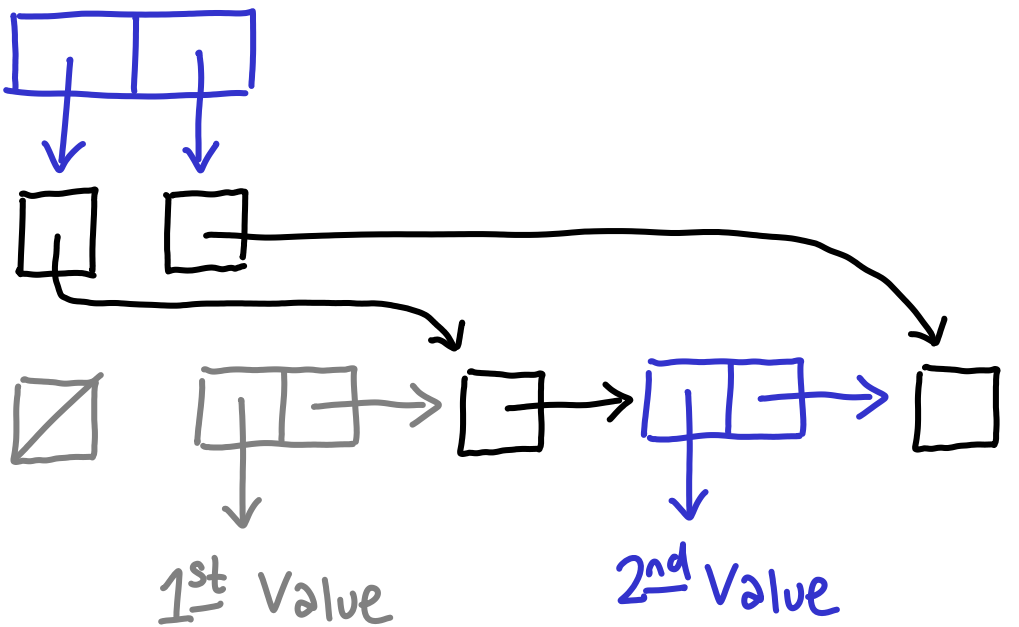
Message Passing: Channels with MVars



Message Passing: Channels with MVars



Message Passing: Channels with MVars



Message Passing

✓ Very simple (basis for many theoretical models; also effective programming style)

for asynchronous messaging

? How big to maintain buffers?

? Unbounded nondeterminism

for synchronous messaging

⚠ Still easy to deadlock

Message passing and immutability

Once you send a message, must not r/w it!

Erlang — separate address space

Haskell — immutable data (but laziness!)

Rust/C++ — ownership types (no aliasing)

like juggling chainsaws

Concurrency and immutability

Immutable data definitionally not shared

Deterministic Parallelism

Concurrency ≠ Parallelism

Deterministic Parallelism

Concurrency

- Availability
- Interactivity

Parallelism

- Speed
- Distribution

Parallelism can be achieved with concurrency

Concurrency without parallelism

Deterministic Parallelism

Goal: Get speedups, w/o headaches of manual synchronization

- MapReduce (similar pipelines)
- Pure computation (sparks)
- Monotonic data structures (LVars)

Software Transactional Memory (Wednesday)

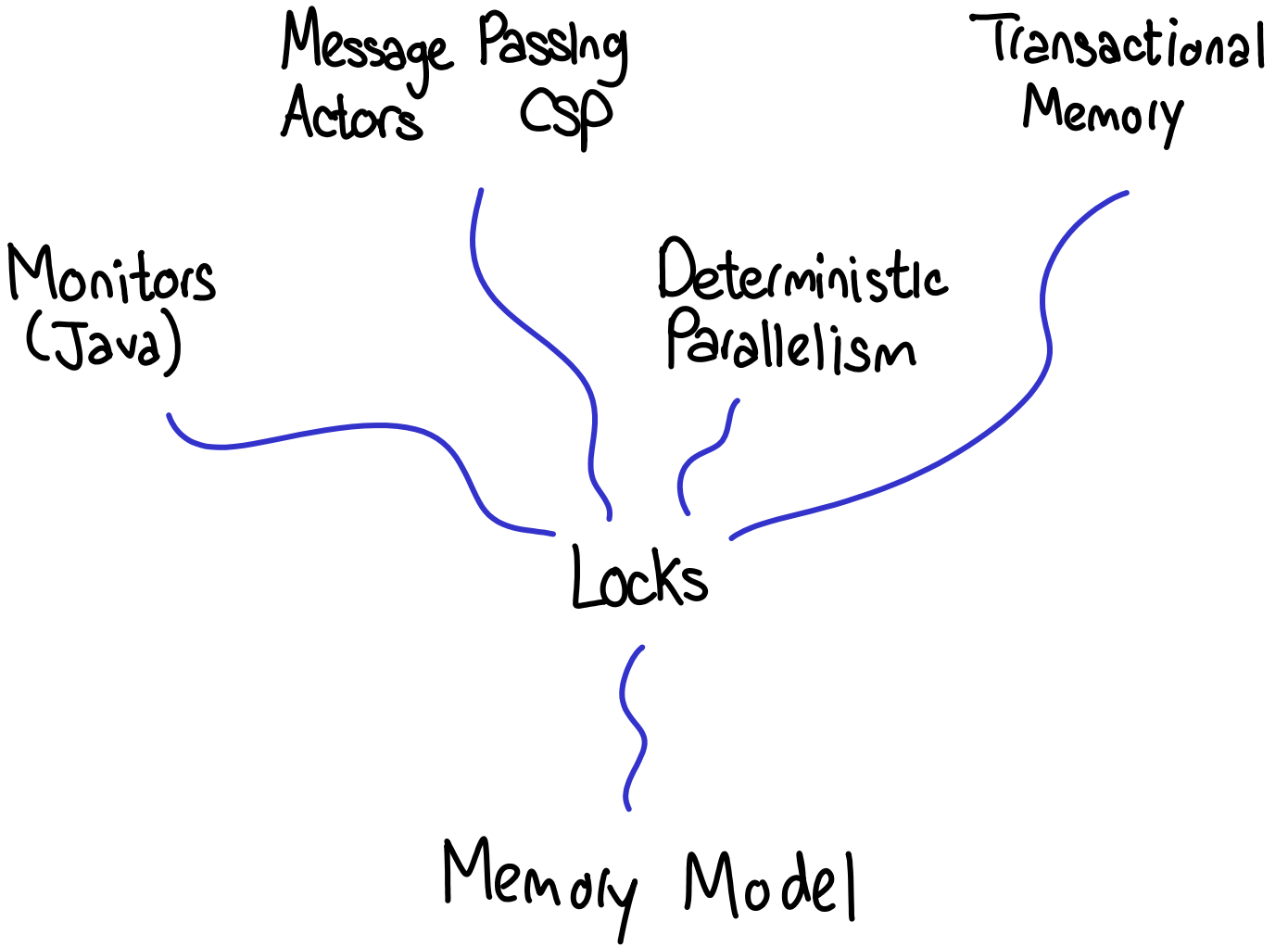
Locks are non-compositional
(nested locking ⇝ deadlock)

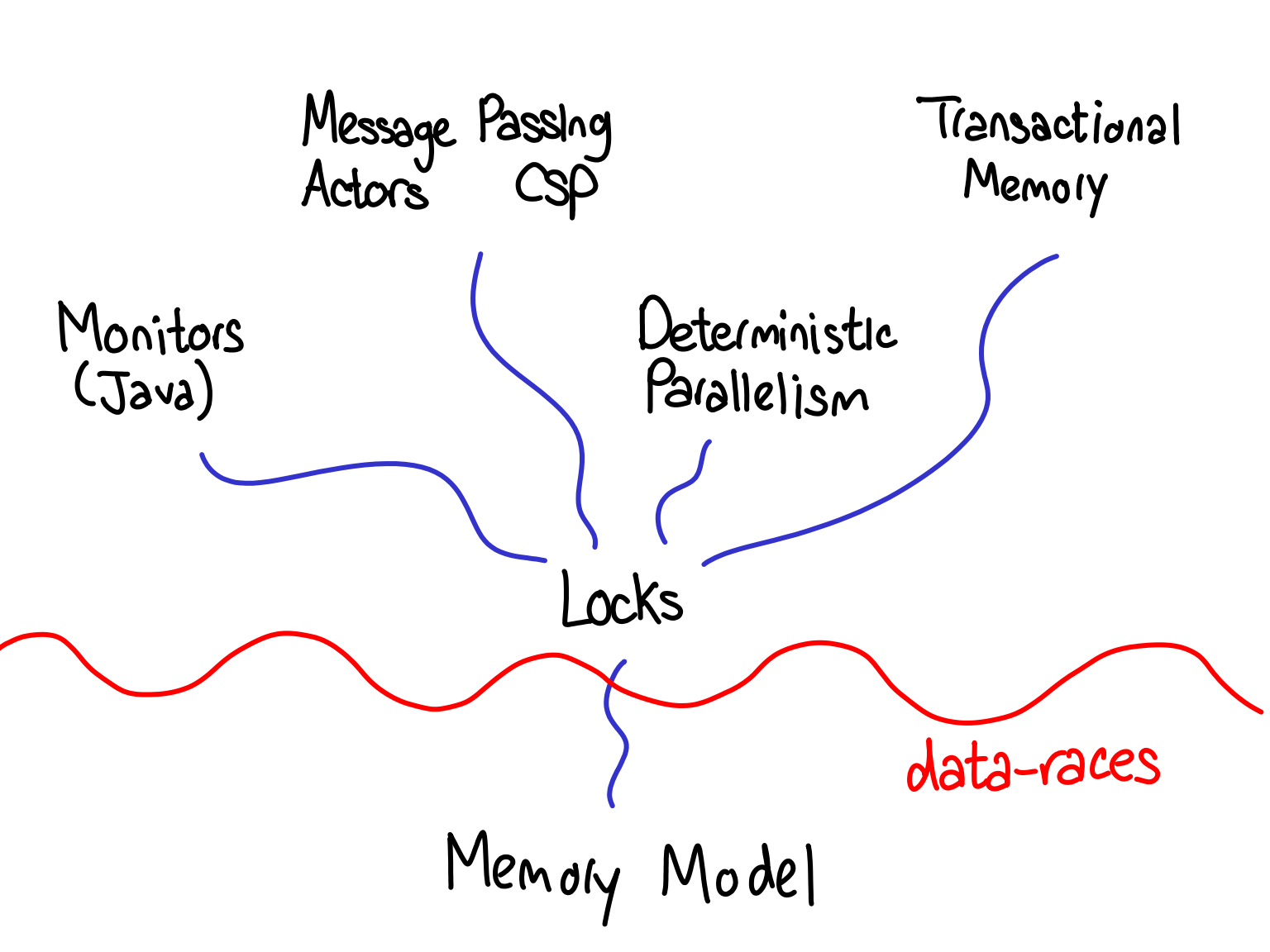
```
synchronized (this) {
    if (balance ≥ 100) {
        balance -= 100;
    } else {
        insufficient funds
    }
}
```

locks

```
atomically {
    if (balance ≥ 100) {
        balance -= 100;
    } else {
        insufficient funds
    }
}
```

behave as if it were an atomic txn


Message Passing
Actors
CSP
Transactional
Memory
Monitors
(Java)
Deterministic
Parallelism
Locks
Memory Model


Message Passing
Actors
CSP
Transactional
Memory
Monitors
(Java)
Deterministic
Parallelism
Locks
data-races
Memory Model

Memory Model

Thread #1	Thread #2
`x = 1;`	`y = 1;`
`tmp1 = y;`	`tmp2 = x;`

What values of tmp1/tmp2 are allowed?

Memory Model

Thread #1	Thread #2
x = 1;	y = 1;
tmp1 = y;	tmp2 = x;

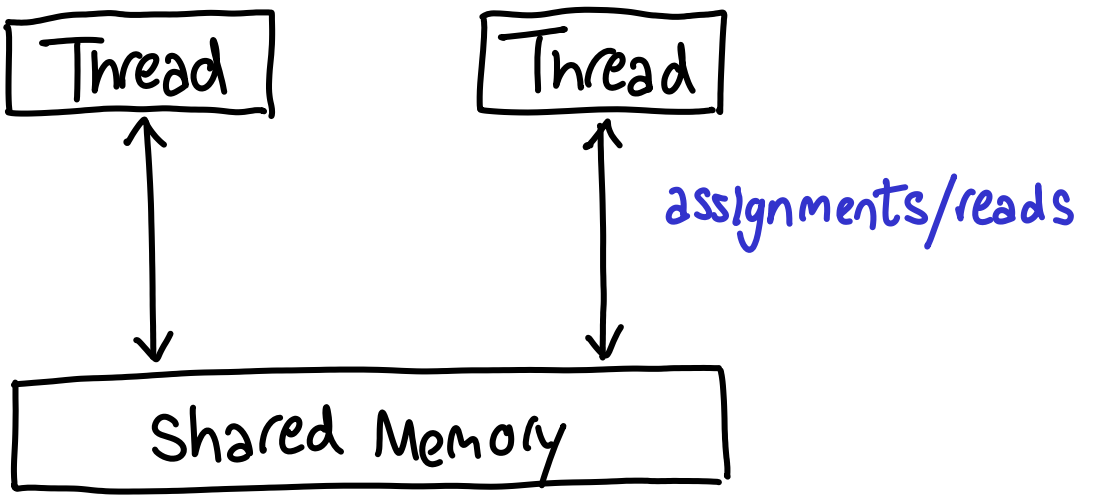
Java has a memory model

Thread #1	Thread #2
MOV [x] ← 1	MOV [y] ← 1
MOV EAX ← [y]	MOV EBX ← [x]

so does your processor

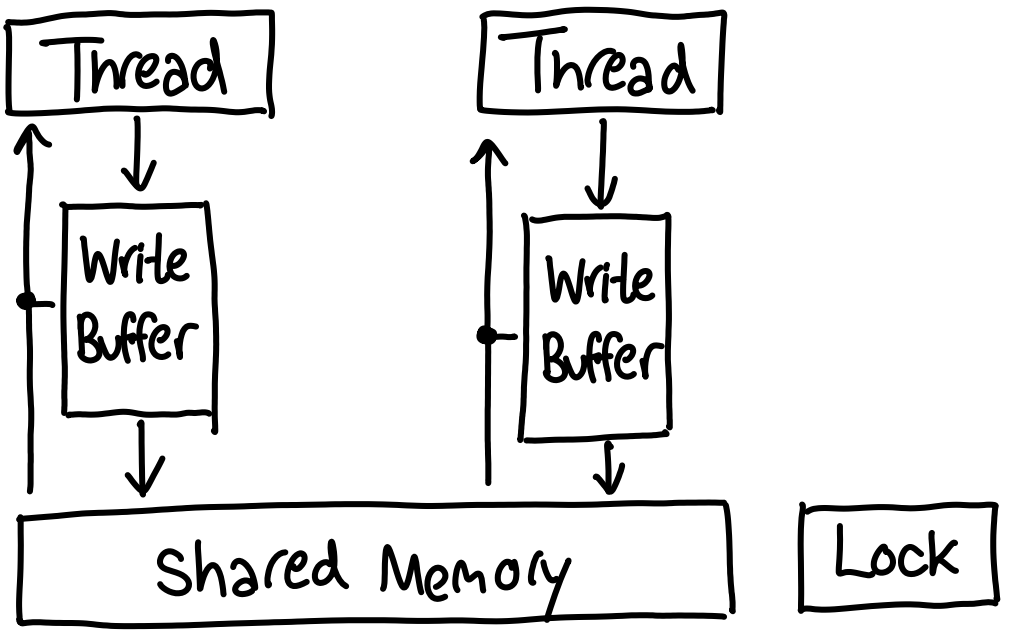
Memory Model: Sequential Consistency

Without data races, should look like this!



(implemented by basically no one)

Memory Model: Total Store Order



SPARC, x86

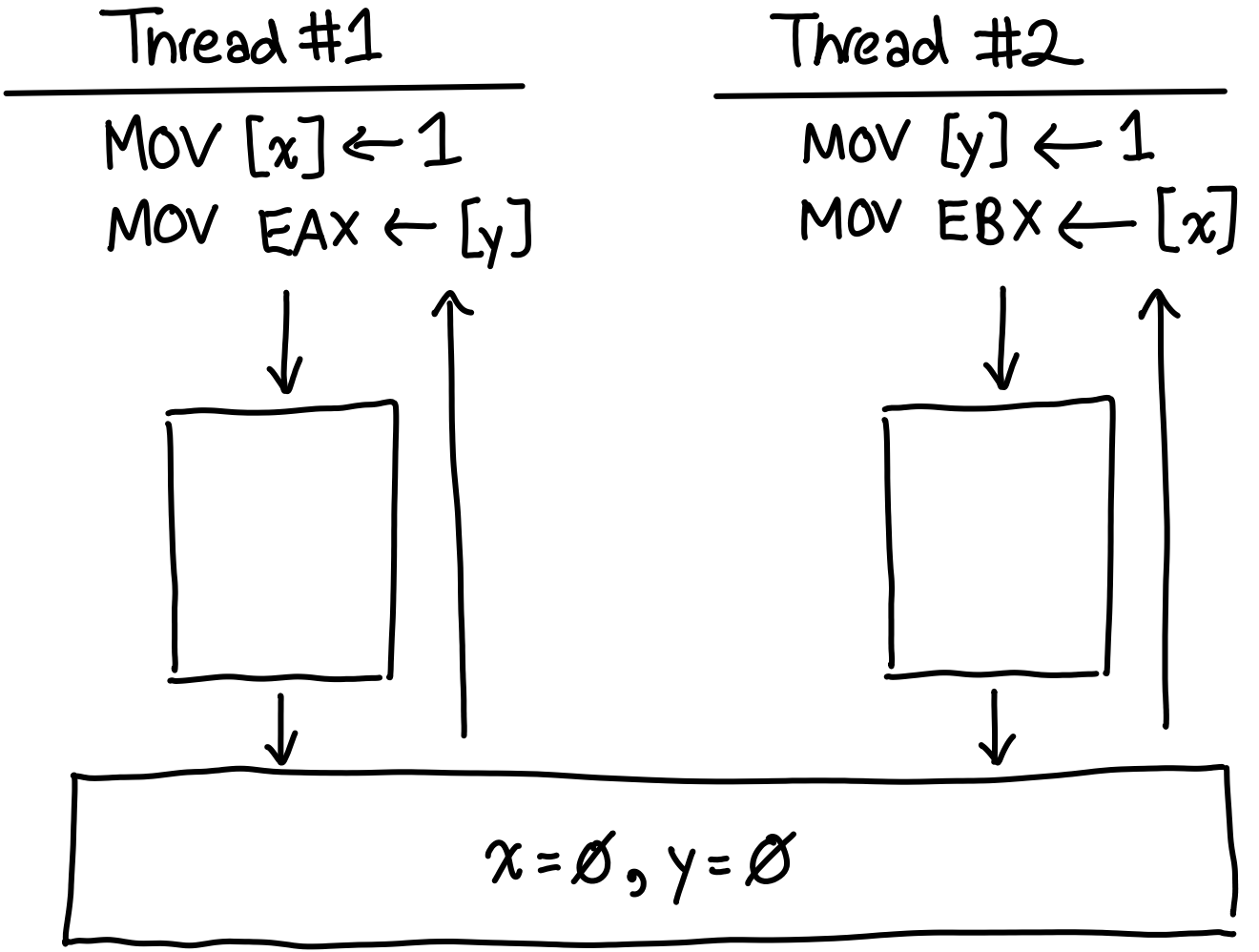
Memory Model: Total Store Order

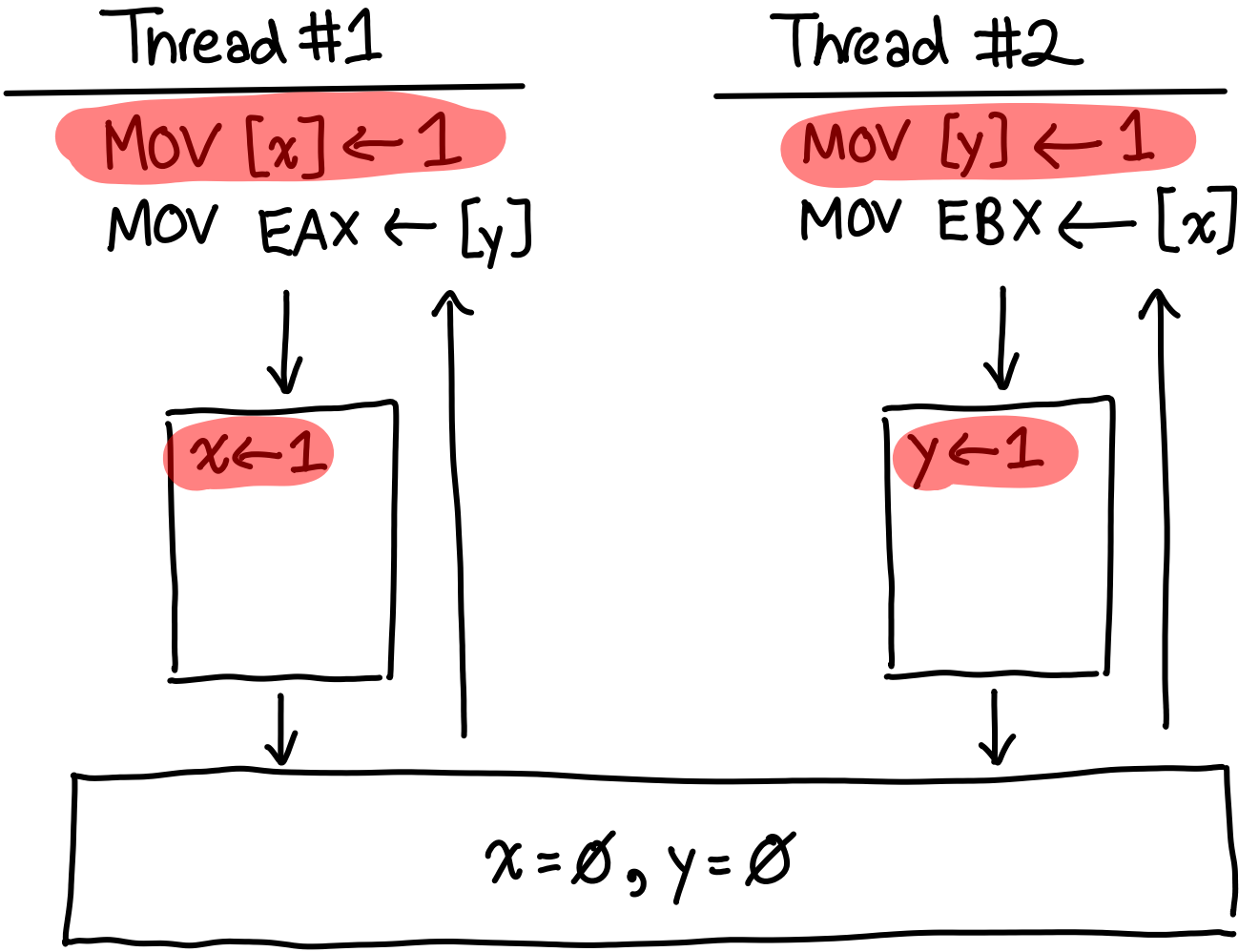
Buffers are FIFO, serve reads if possible

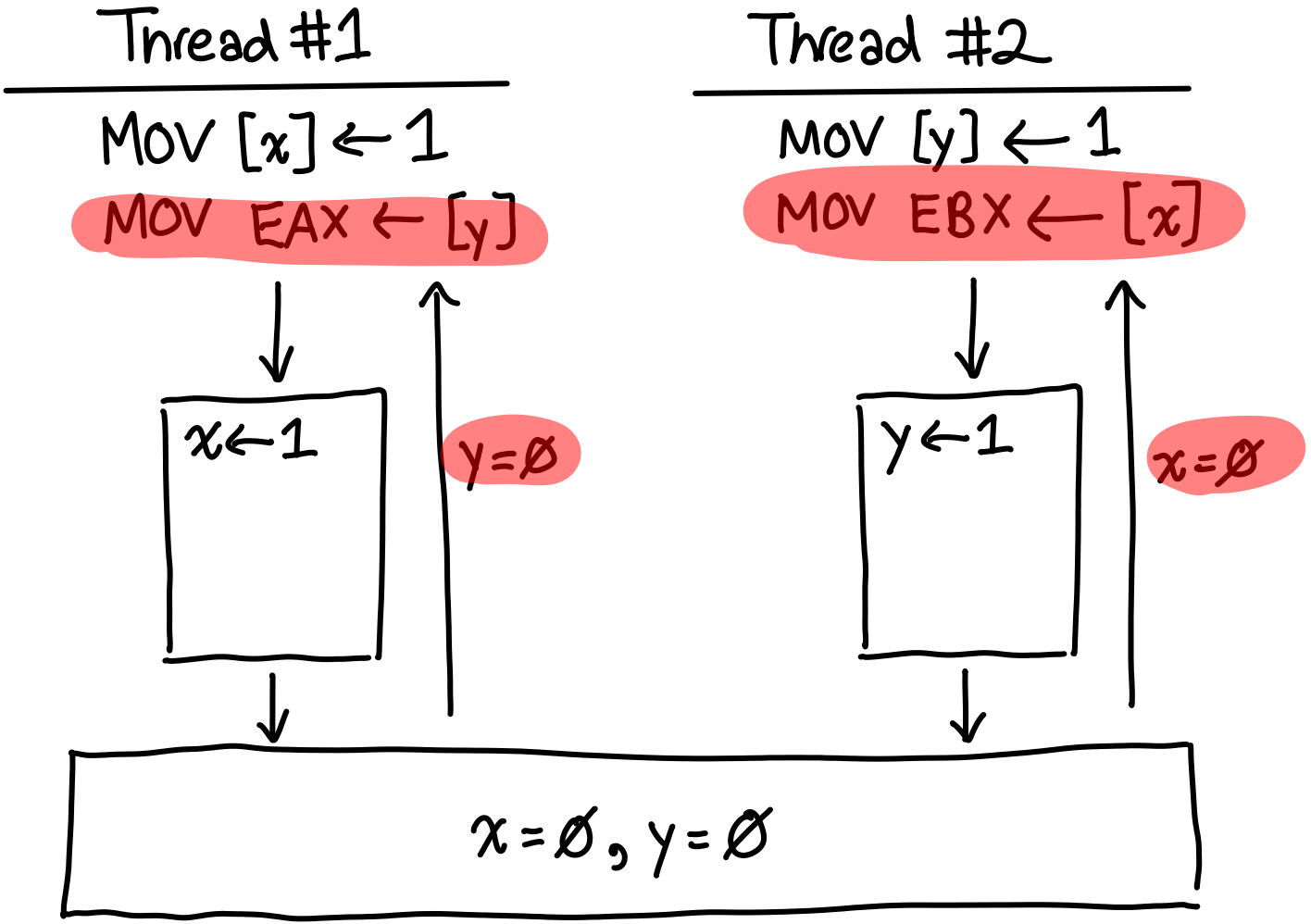
MFENCE flushes buffer

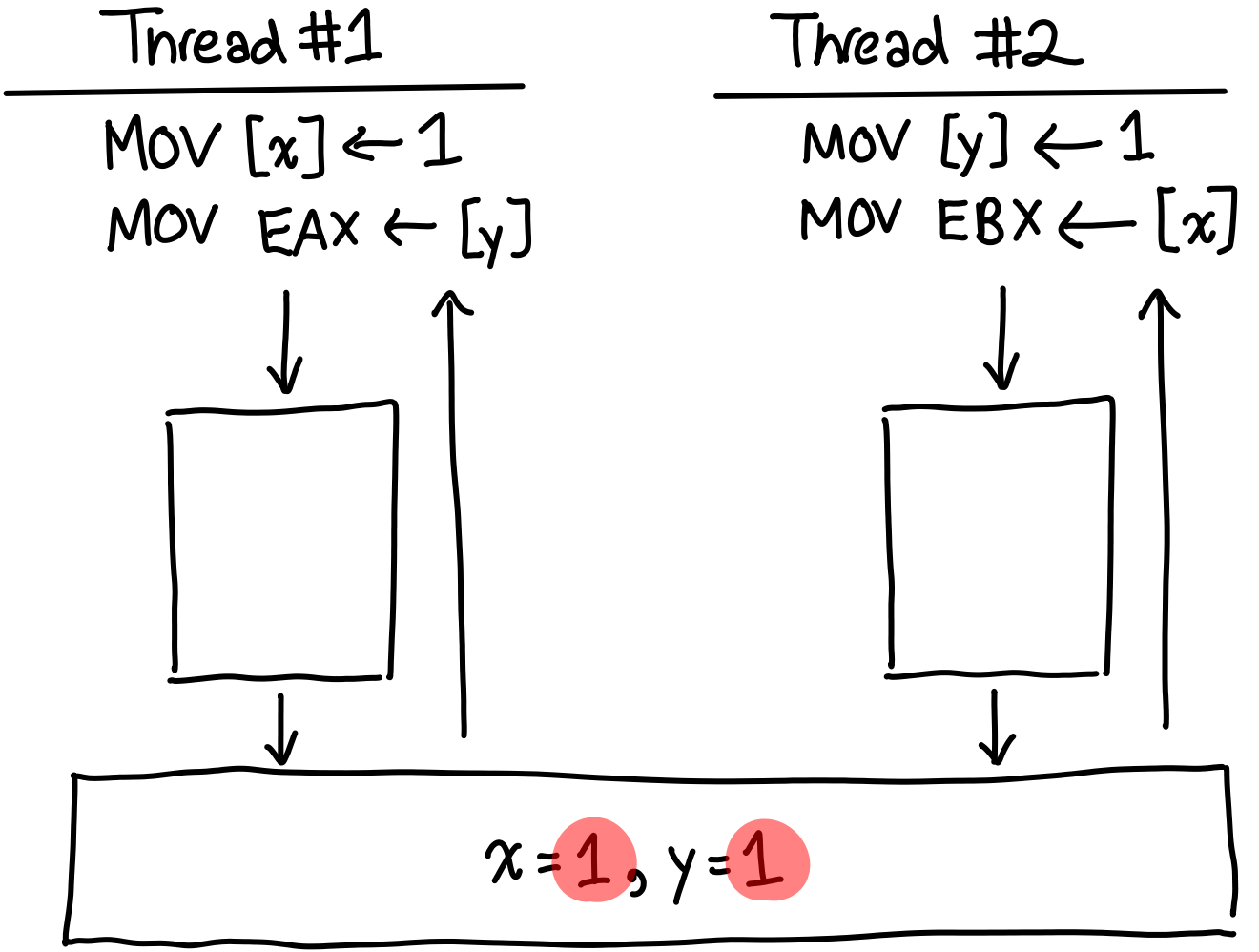
LOCK takes global lock, flushes buffer at end
(e.g. LOCKADD, XCHG)

Lock blocks buffered writes


Thread #1
MOV [x] ← 1
MOV EAX ← [y]
Thread #2
MOV [y] ← 1
MOV EBX ← [x]
x = ∅, y = ∅


Thread #1
MOV [x] ← 1
MOV EAX ← [y]
x ← 1
Thread #2
MOV [y] ← 1
MOV EBX ← [x]
y ← 1
x = ∅, y = ∅


Thread #1
MOV [x] ← 1
MOV EAX ← [y]
x ← 1
y = ∅
Thread #2
MOV [y] ← 1
MOV EBX ← [x]
y ← 1
x = ∅
x = ∅, y = ∅


Thread #1
MOV [x] ← 1
MOV EAX ← [y]
Thread #2
MOV [y] ← 1
MOV EBX ← [x]
x = 1, y = 1

On entry the address of spinlock is in register EAX and the spinlock is unlocked iff its value is 1			
acquire:	LOCK;DEC	[EAX]	; LOCK'd decrement of [EAX]
	JNS	enter	; branch if [EAX] was ≥ 1
spin:	CMP	[EAX],0	; test [EAX]
	JLE	spin	; branch if [EAX] was ≤ 0
	JMP	acquire	; try again
enter:	; the critical section starts here		
release:	MOV	[EAX]←1	

does this need to be locked?

Memory Model: ARM/POWER

- May read/write out of order
- Writes may not become visible to all threads at the same time point

Memory Model: Java

Java Language

Hardware Architecture

```
x = 1;
tmp1 = y;
```

```
MOV [x] ← 1
MOV EAX ← [y]
```

Memory Model: Java

Java Language

Hardware Architecture

```
x = 1;
tmp1 = y;
```

optimizations!

```
MOV EAX ← [y]
MOV [x] ← 1
```

Memory Model: Java

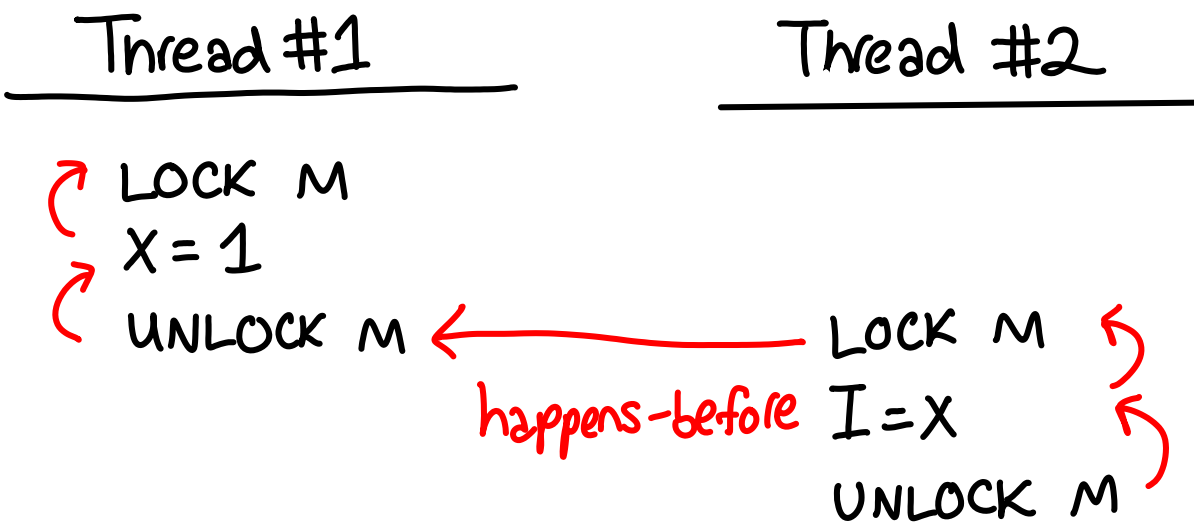
- What optimizations are acceptable?
  - Notice, concurrency breaks abstraction barrier relied upon by optimization
- What program outputs can occur?
  - Undefined is unacceptable if it leads to a security problem

Memory Model: Java

Happens-before

Memory Model: Java

Happens-before (Synchronizes-with)

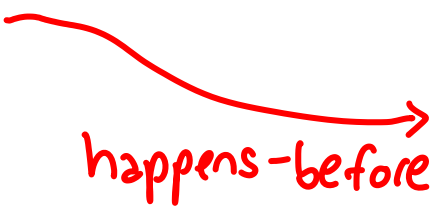


Memory Model: Java

Happens-before (Synchronizes-with)

volatile x;

Thread #1	Thread #2
x = 1;	
	y = x;



(Not like C volatiles!)

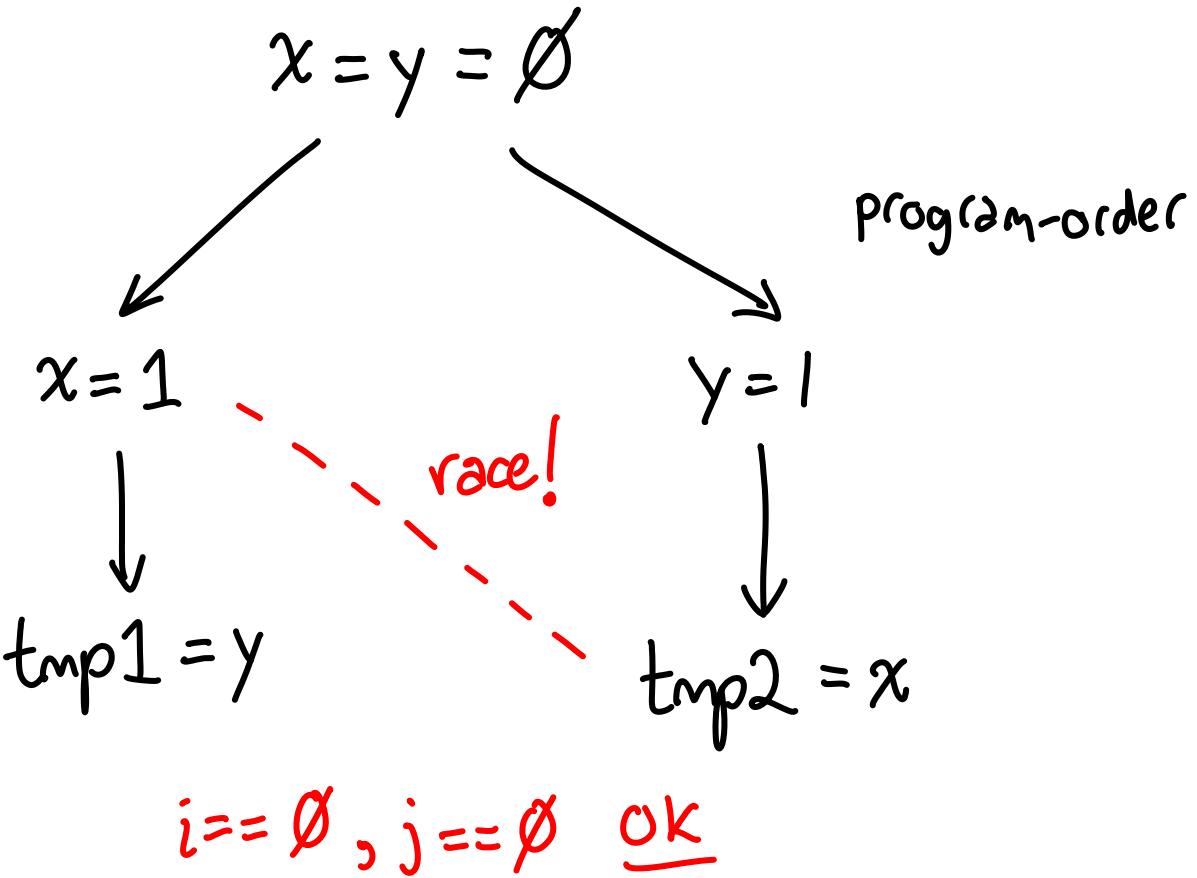
Memory Model: Java

Happens-before:

Program order + Synchronizes with

Data-race: two accesses on different threads (one being a write) not ordered by happens-before

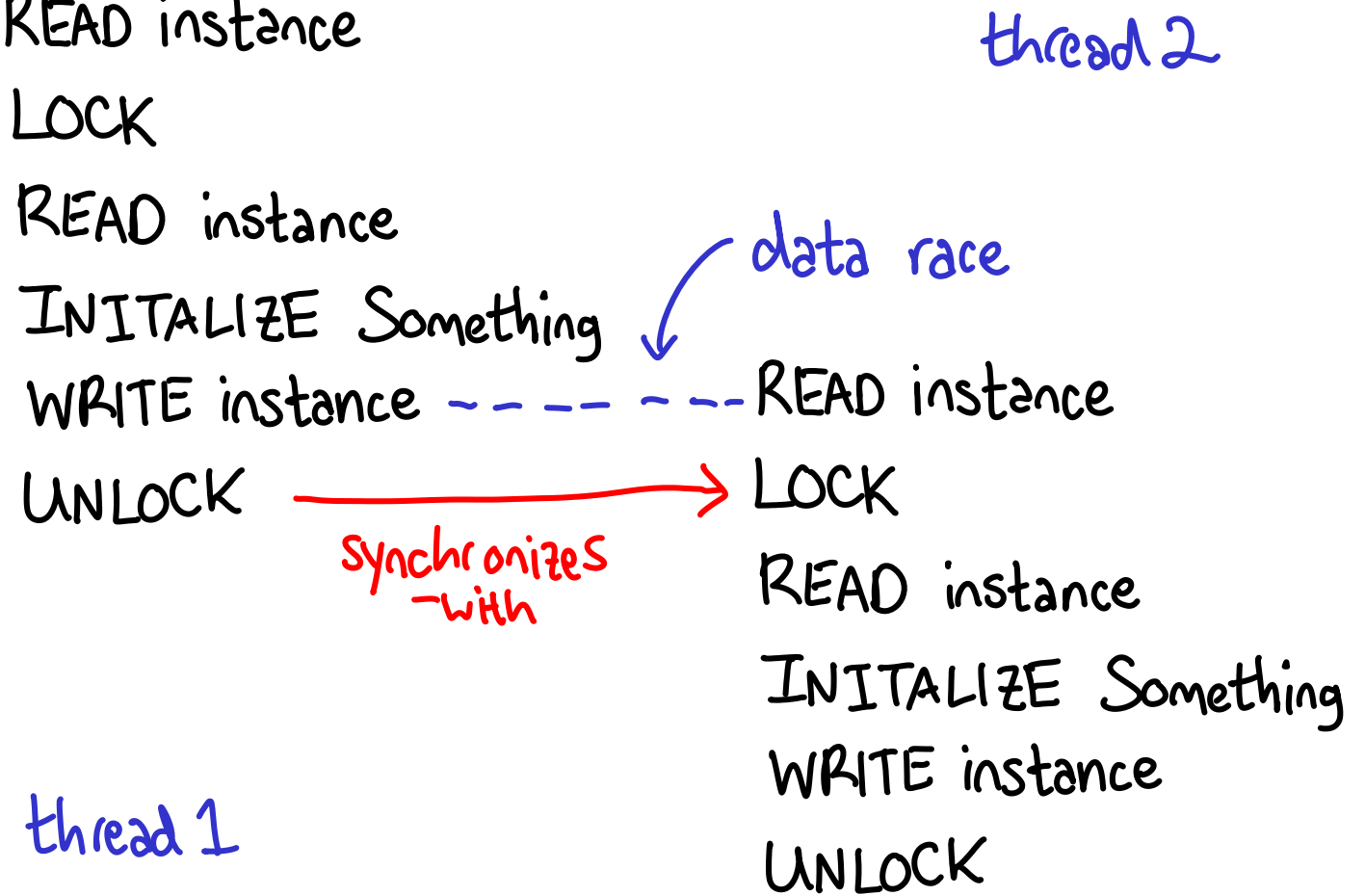
If no data-races, program acts as if it is sequentially consistent (if not? don't ask!)


x = y = ∅
program-order
x = 1
y = 1
race!
tmp1 = y
tmp2 = x
i == ∅ , j == ∅ ok

Double-checked locking

```
private static Something instance = null;

public Something getInstance() {
    if (instance == null) {
        synchronized (this) {
            if (instance == null) {
                instance = new Something();
            }
        }
    }
    return instance;
}
```

Double-checked locking



Double-checked locking

READ instance

LOCK

READ instance

WRITE instance

INITALIZE Something

UNLOCK

reorder!

thread 2

reads the uninitialized object

READ instance

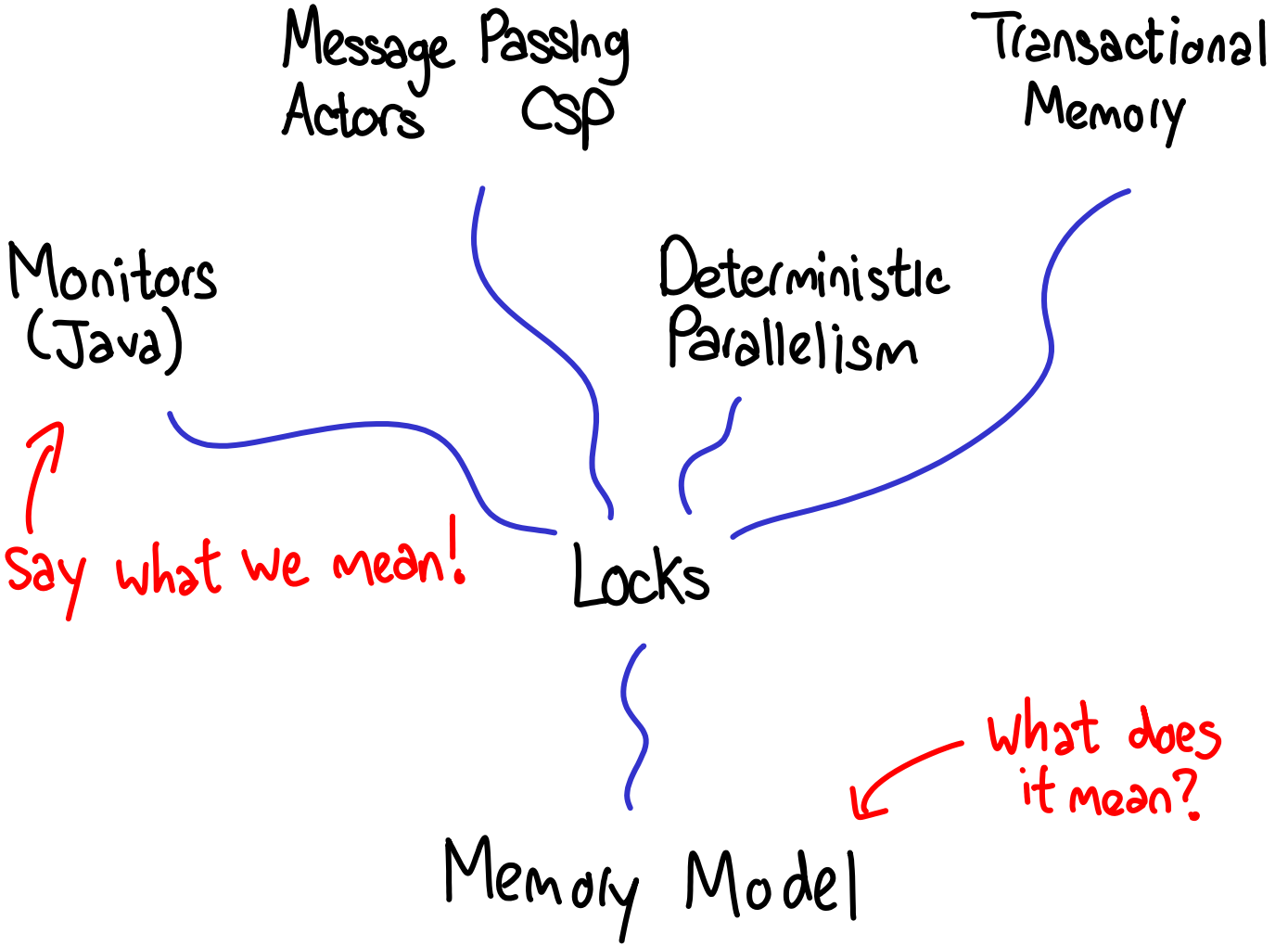
thread 1

Recall

How can programming languages make concurrent programming easier?

What abstractions are most effective?

Wednesday: Forward-looking research ideas


Message Passing
Actors
CSP
Transactional Memory
Monitors (Java)
Deterministic Parallelism
Say what we mean!
Locks
What does it mean?
Memory Model

Extra examples

- Implementing monitors w/ MVars