Buffer & File Optimization

Introduction to Databases

DataLab

CS, NTHU

Outline

- Useful Java Classes for Concurrency
- Lock Striping
- Summary of File & Buffer Optimization

Outline

- **Useful Java Classes for Concurrency**
- Lock Striping
- Summary of File & Buffer Optimization

ReentrantLock

- An implementation of `Lock`
  - Provided in `java.util.concurrent.locks`
- A `ReentrantLock` has better performance than a `synchronized` block in multi-threading scenario

```java
class X {
    private final ReentrantLock lock = new ReentrantLock();

    public void m() {
        lock.lock(); // block until condition holds
        try {
            // do something
        } finally {
            lock.unlock();
        }
    }
}
```

- See more [here](#)

ReentrantReadWriteLock

- An implementation of `ReadWriteLock`
  - Provided in `java.util.concurrent.locks`

- In addition to all functions `ReentrantLock` provide, `ReentrantReadWriteLock` **also have** `ReadLock` **and** `WriteLock`
  - A thread will be blocked during acquiring a `ReadLock` only if there is another thread holds a `WriteLock`

- See more [here](#)

ReentrantReadWriteLock

```java
class Counter {
    // Locks
    private final ReentrantReadWriteLock rwLock = new ReentrantReadWriteLock();
    private final Lock rLock = rwLock.readLock();
    private final Lock wLock = rwLock.writeLock();

    // Value
    private int value = 0;

    public int get() {
        rLock.lock();
        try {
            return value;
        } finally {
            rLock.unlock();
        }
    }

    public void increment() {
        wLock.lock();
        try {
            value += 1;
        } finally {
            wLock.unlock();
        }
    }
}
```
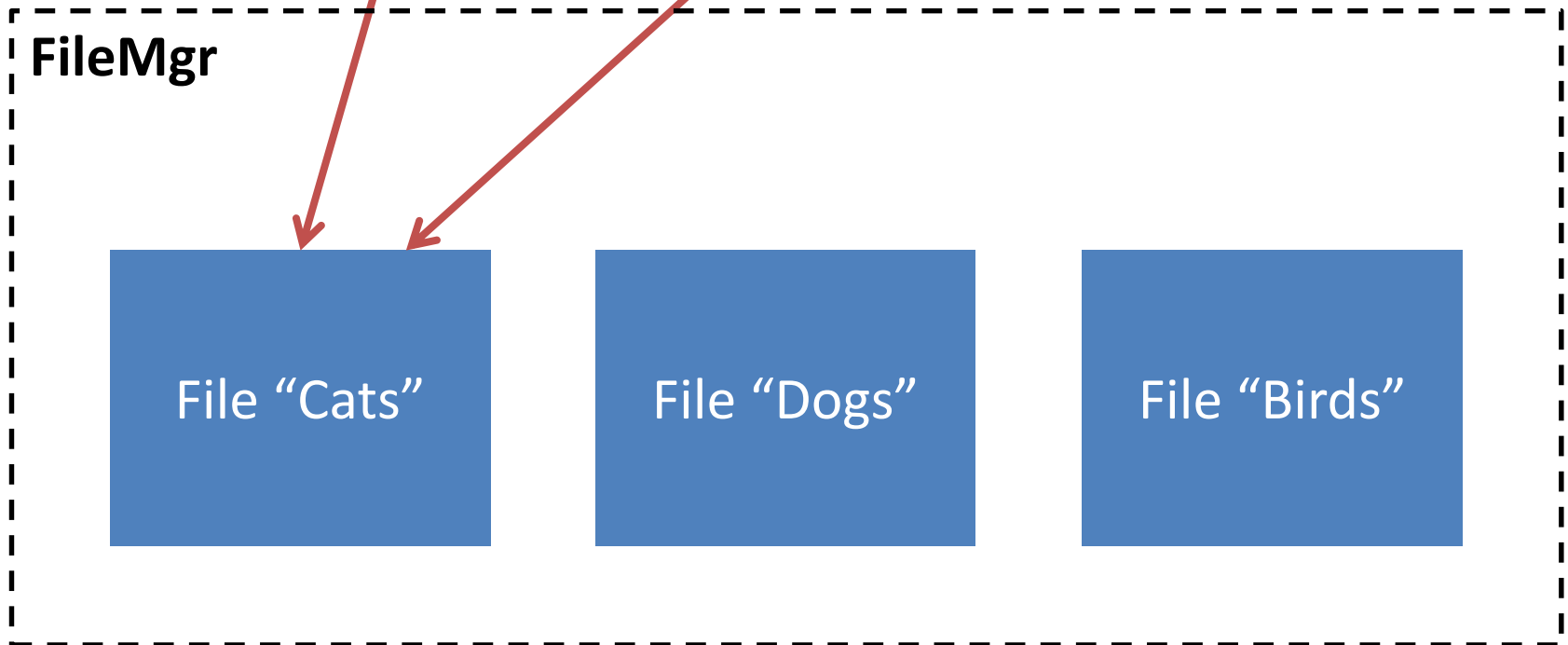
ConcurrentHashMap

- A thread-safe `HashMap`
  - Provided in `java.util.concurrent`
- A `ConcurrentHashMap` works better than a synchronized `HashMap` which is just simply protected by synchronized blocks
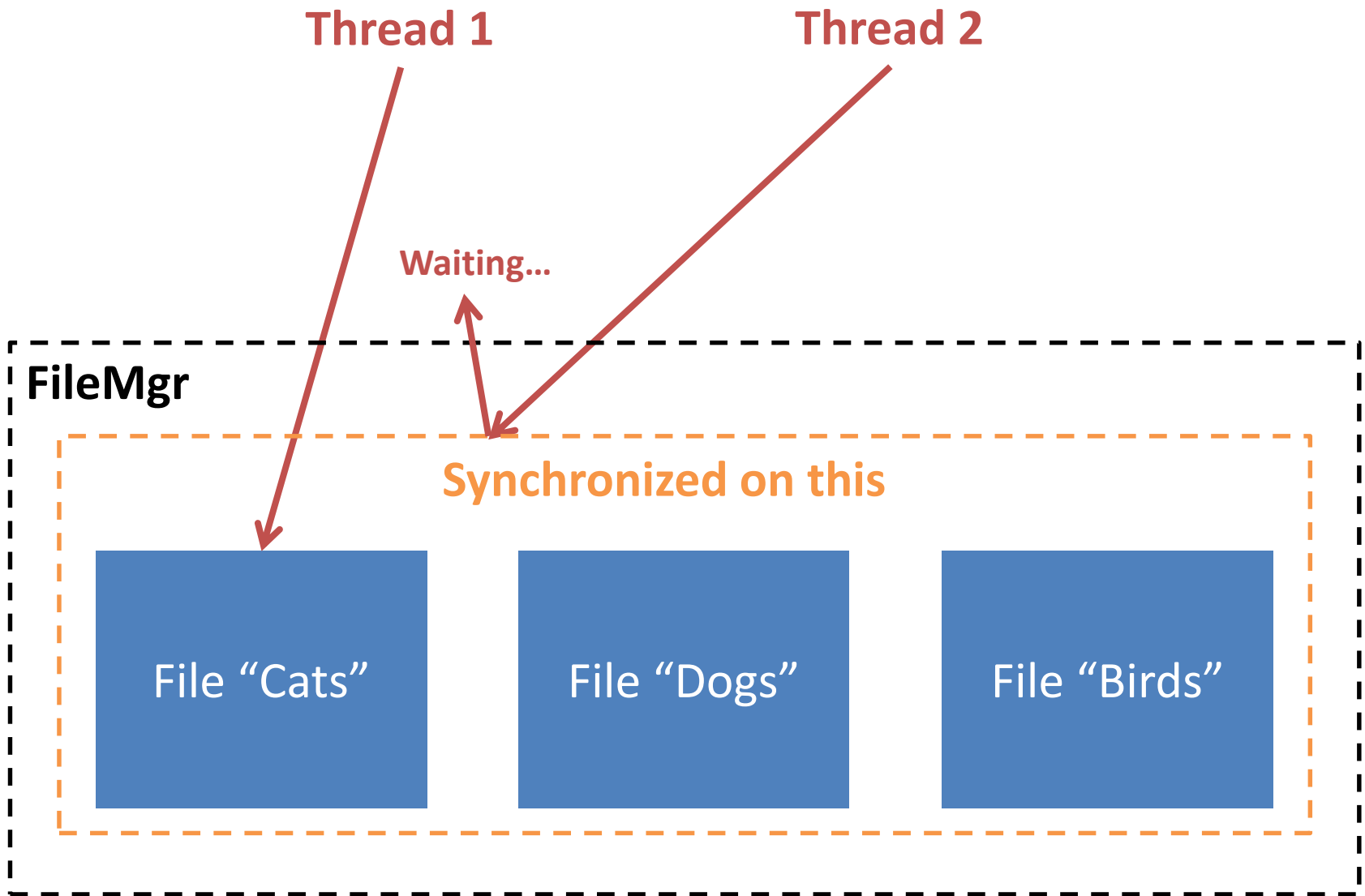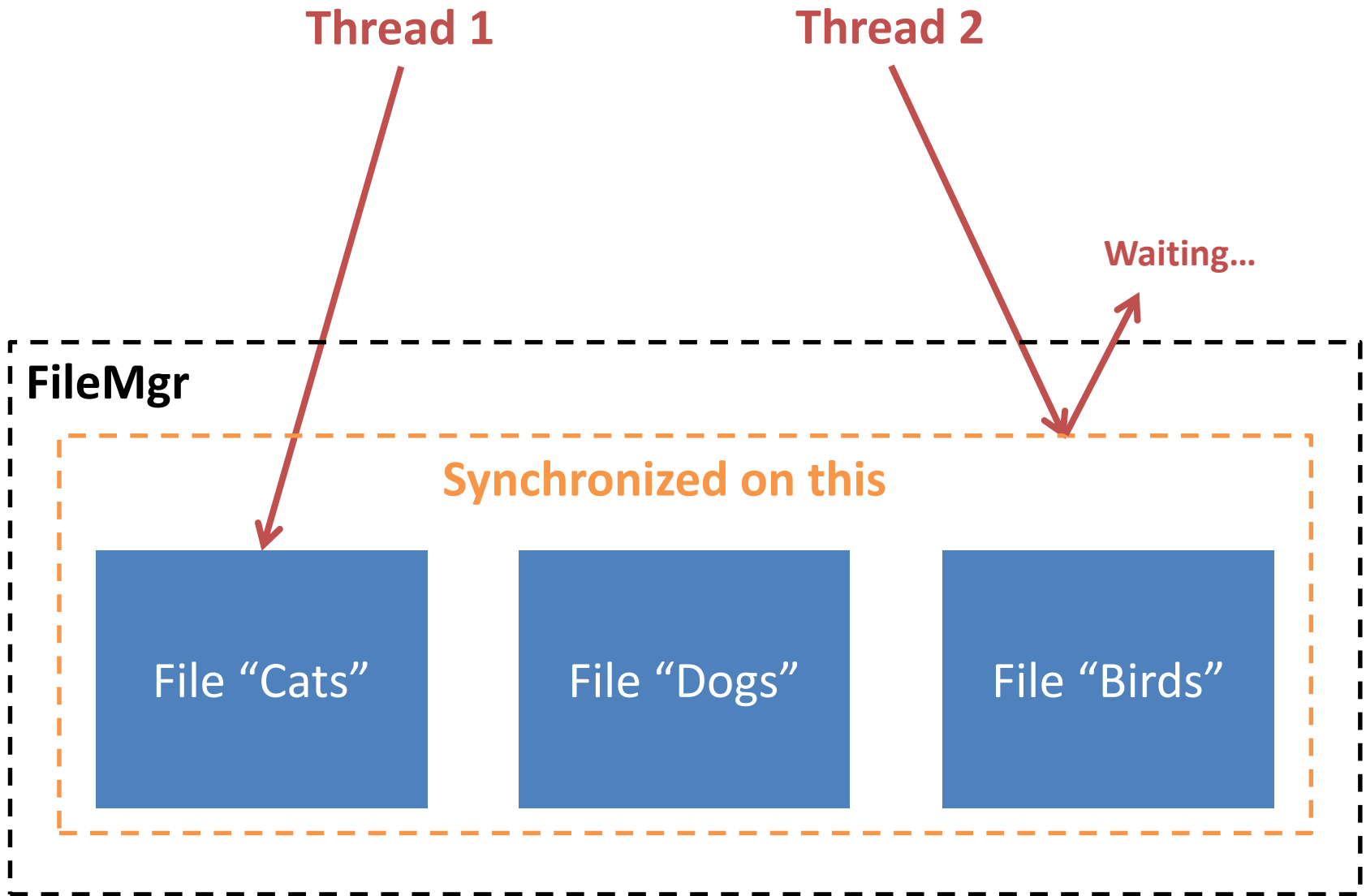
- See more [here](#)

Outline

- Useful Java Classes for Concurrency
- **Lock Striping**
- Summary of File & Buffer Optimization
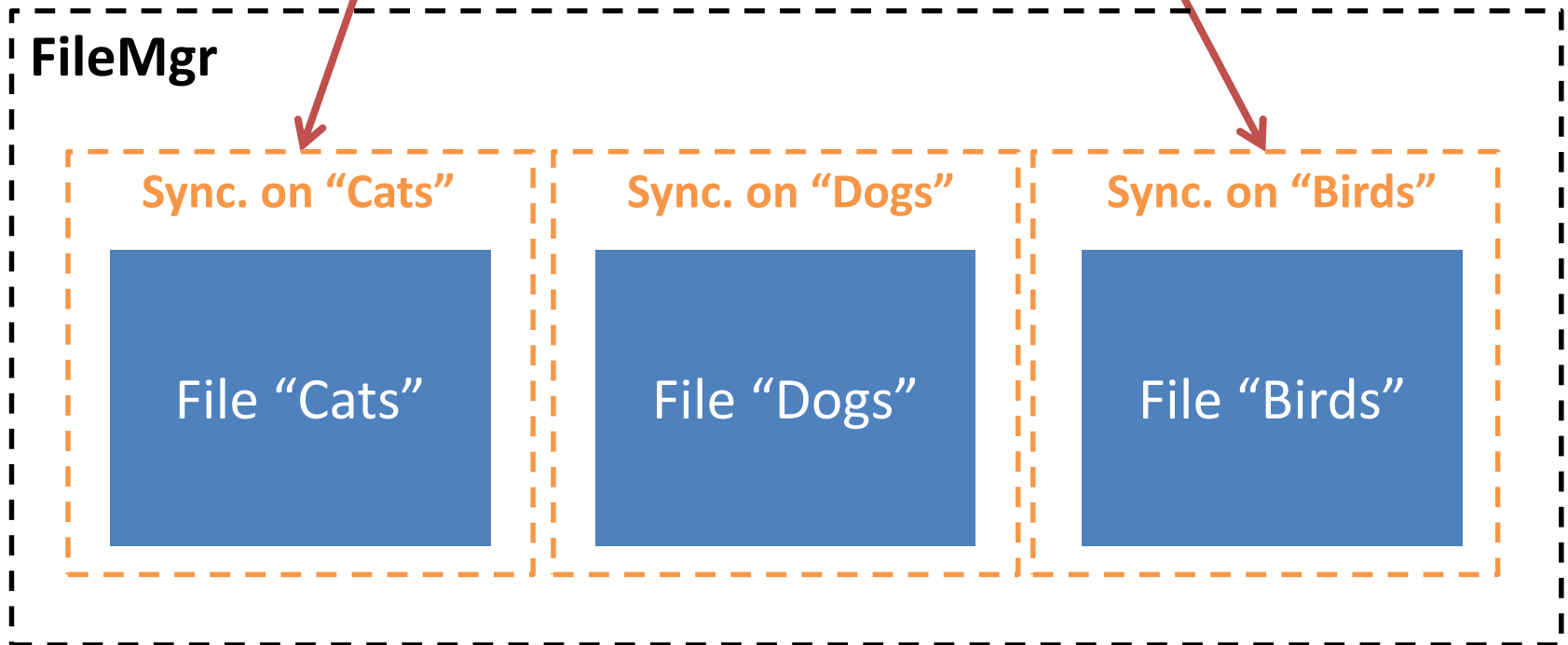
Thread 1 Thread 2

Waiting...

FileMgr

Synchronized on this

File "Cats"   File "Dogs"   File "Birds"

Thread 1

Thread 2

FileMgr

Sync. on "Cats"

Sync. on "Dogs"

Sync. on "Birds"

File "Cats"

File "Dogs"

File "Birds"

Global Synchronization

```java
class ResourceMgr {

    private Map<String, Resource> resourcePool =
            new HashMap<String, Resource>();

    public synchronized void doSomething(String key) {
        Resource res = getResource(key);
        res.doAThing();
    }

    private Resource getResource(String key) {
        Resource res = resourcePool.get(key);

        if (res == null) {
            res = new Resource();
            resourcePool.put(key, res);
        }

        return res;
    }
}
```

Synchronization on Each Resource Object

```java
class ResourceMgr {

    private Map<String, Resource> resourcePool =
            new HashMap<String, Resource>();

    public void doSomething(String key) {
        Resource res = getResource(key);

        synchronized (res) {
            res.doAThing();
        }
    }
```

Lock on the required object

```java
    private Resource getResource(String key) {
        Resource res = resourcePool.get(key);

        if (res == null) {
            res = new Resource();
            resourcePool.put(key, res);
        }

        return res;
    }
}
```

There is a problem here

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```

Pool

Thread 1 key="meow" res = NULL

Thread 2 key="meow" res = NULL

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```
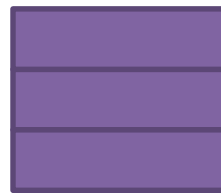
Resource 1

Pool

Thread 1   **key="meow"**   **res** = **NULL**

Thread 2   **key="meow"**   **res** = **NULL**

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```

Pool

meow

Resource 1

Thread 1 **key="meow"** **res**

Thread 2 **key="meow"** **res** = NULL

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```
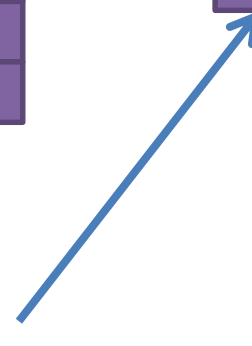
Pool

Resource 1

Resource 2

meow

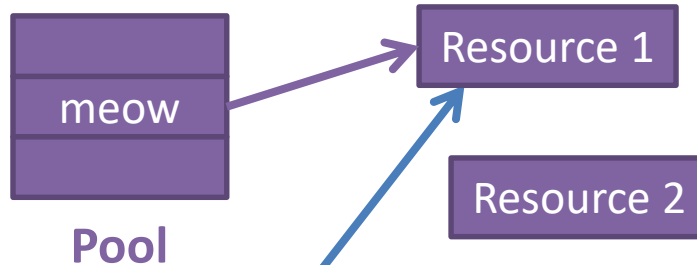**Thread 1**   **key="meow"**   **res**

Thread 2   **key="meow"**   **res** = NULL

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```

Pool

Resource 1

Resource 2

meow

Thread 1   **key="meow"**   **res**

Thread 2   **key="meow"**   **res**

Race Condition

```
private Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;

}
```
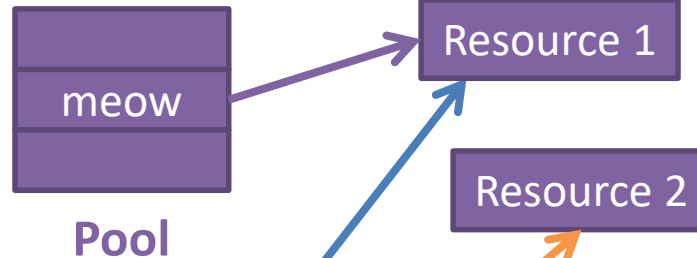
Pool

meow

Resource 1

Resource 2

Thread 1   key="meow"   **res**

Thread 2   key="meow"   **res**

There are two resource with the same key !! And only 1 can be found

Solution

The problem solved, but not good enough

```java
private synchronized Resource getResource(String key) {
    Resource res = resourcePool.get(key);

    if (res == null) {
        res = new Resource();
        resourcePool.put(key, res);
    }

    return res;
}
```
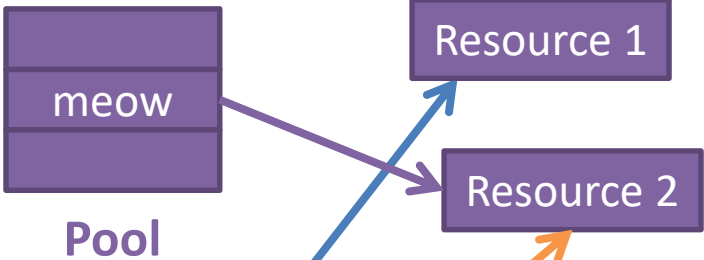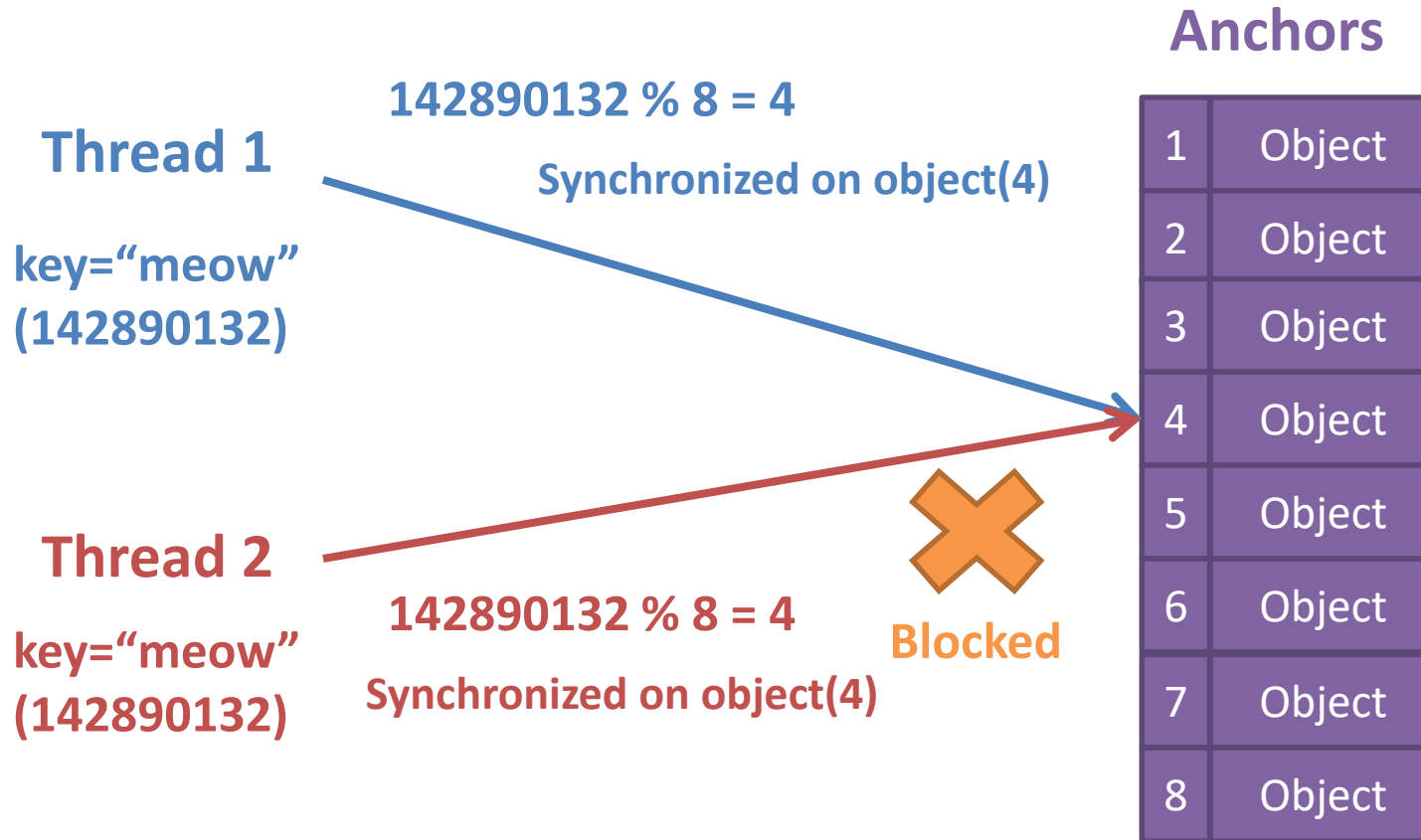
Lock Striping

- Lock striping basically uses a fixed-size, shared collection of locks to reduce the contention on the same object

Lock Striping

Anchors

142890132 % 8 = 4

Thread 1

Synchronized on object(4)

key="meow"
(142890132)

Thread 2

142890132 % 8 = 4

key="meow"
(142890132)

Synchronized on object(4)

Blocked

1	Object
2	Object
3	Object
4	Object
5	Object
6	Object
7	Object
8	Object

Lock Striping

Anchors

Thread 1

142890132 % 8 = 4

Synchronized on object(4)

key="meow"
(142890132)

Thread 2

Synchronized on object(6)

key="kerker"
(453621342)

453621342 % 8 = 6

Pass

1	Object
2	Object
3	Object
4	Object
5	Object
6	Object
7	Object
8	Object

Final Solution

```java
private Object[] anchors = new Object[100];

private Object getAnchor(String key) {
    return anchors[key.hashCode() % anchors.length];
}

private Resource getResource(String key) {
    synchronized (getAnchor(key)) {
        Resource res = resourcePool.get(key);

        if (res == null) {
            res = new Resource();
            resourcePool.put(key, res);
        }

        return res;
    }
}
```
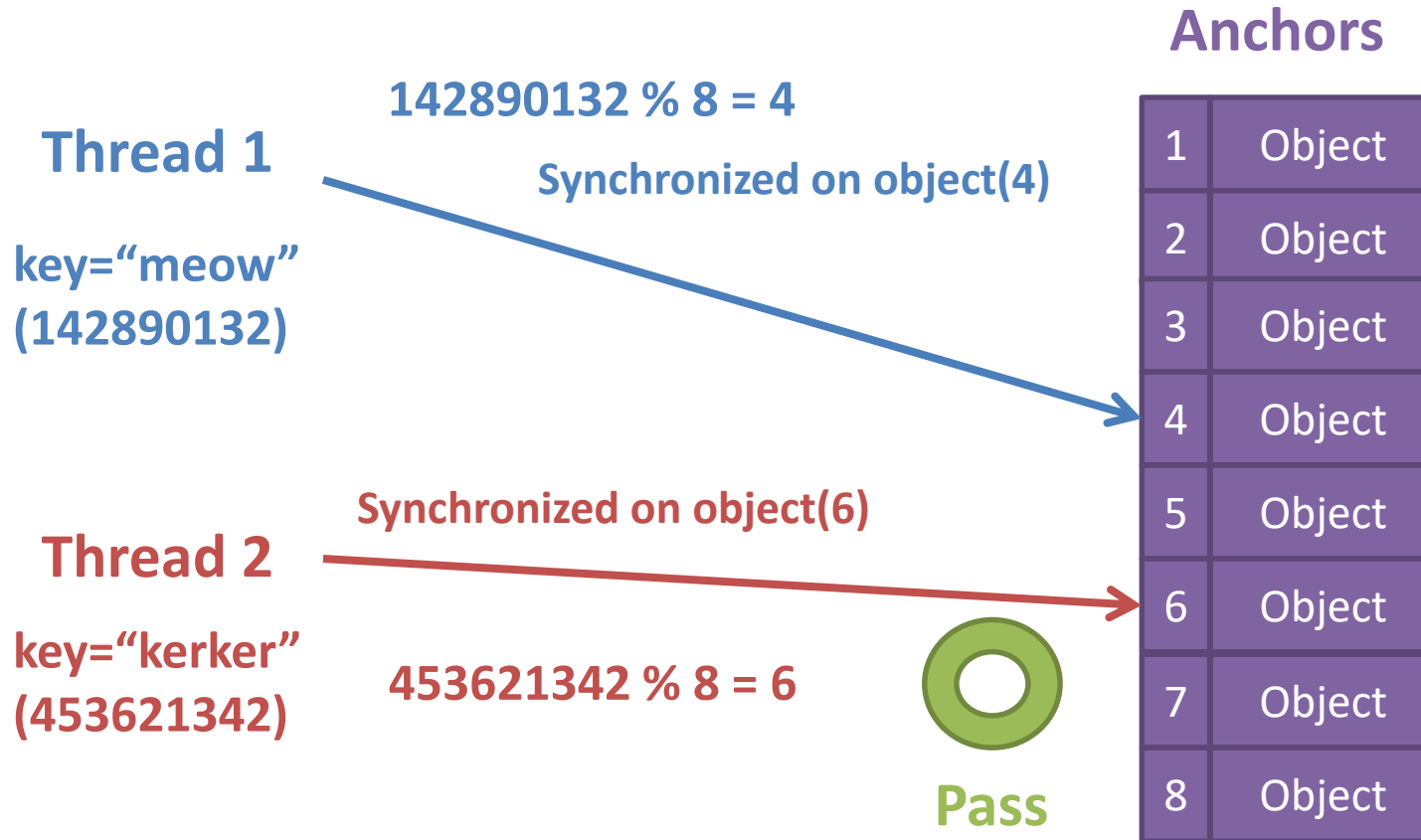
Don't forget to use ConcurrentHashMap for resource pool

Outline

- Useful Java Classes for Concurrency
- Lock Striping
- **Summary of File & Buffer Optimization**

File Optimization

- Read Write Lock

  – We use `RentreenReadWriteLock` in each `IoChannel`, use `ReadLock` for reading and `WriteLock` for modifications

- Lock Striping

  – Use lock-striping in `getFileChannel()`

- Caching

  – Cache the hashcode of `BlockId`

Buffer Optimization

- Reduce the size of critical section as small as possible
  - e.g. `BufferMgr.pin()` and `pinNew()`
- Read Write Lock
  - For each `Buffer`
- Lock Striping
  - In `BufferPoolMgr.pin()` and `pinNew()`

Some Research on `pin()`

- According to a research [1], txs usually take more time in buffer manager than in other modules
- Some researchers of HP lab found `pin()` is a big bottleneck when traversing B-tree indexes [2]
  - They purposed a new way to optimize buffer manager for B-tree indexes
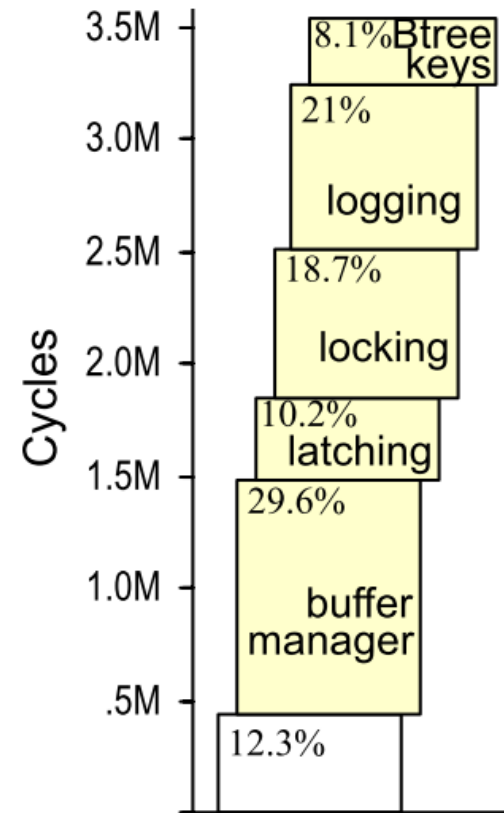
[1] "OLTP Through the Looking Glass, and What We Found There." in SIGMOD'08
[2] "In-Memory Performance for Big Data" in VLDB'14