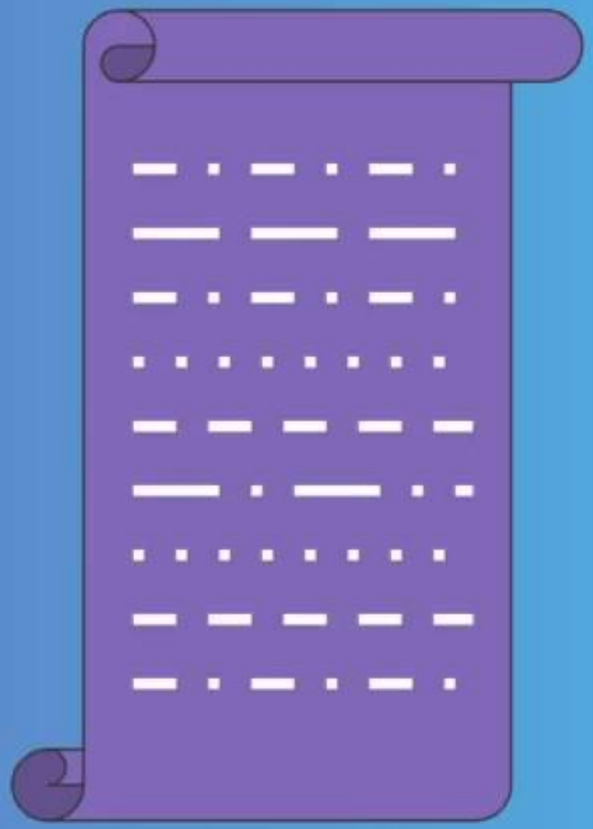
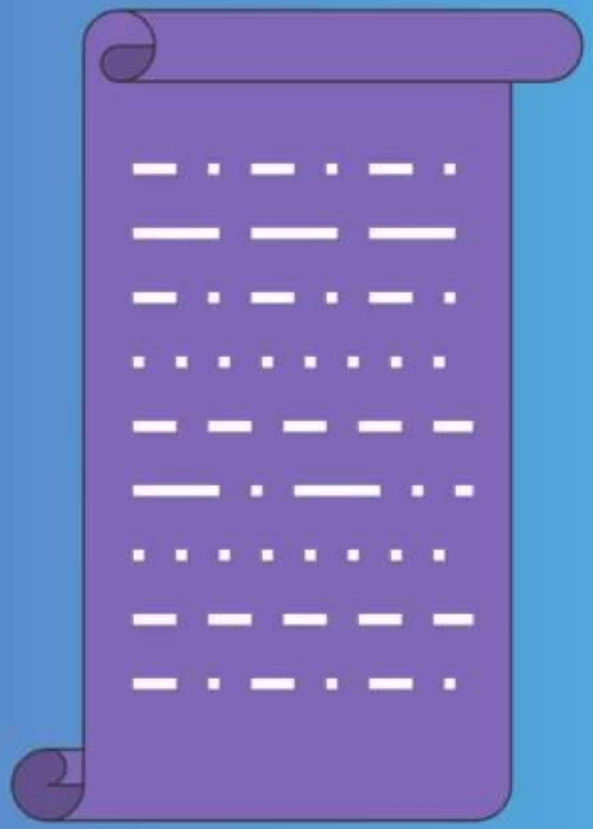
Java Memory Model



Java Memory Model



Out of order execution

Performance driven changes

done by

Compiler, JVM or CPU

Out of order execution

Performance driven changes

done by

Compiler, JVM or CPU

```
a = 3;
b = 2;
a = a + 1;
```

Instructions

- → Load a
- → Set to 3
- → Store a

- → Load b
- → Set to 2
- → Store b

- → Load a
- → Set to 4
- → Store a

Consider this code

```
a = 3;
a = a + 1;

b = 2;
```

Instructions

- → Load a
- → Set to 3
- → Set to 4
- → Store a

- → Load b
- → Set to 2
- → Store b

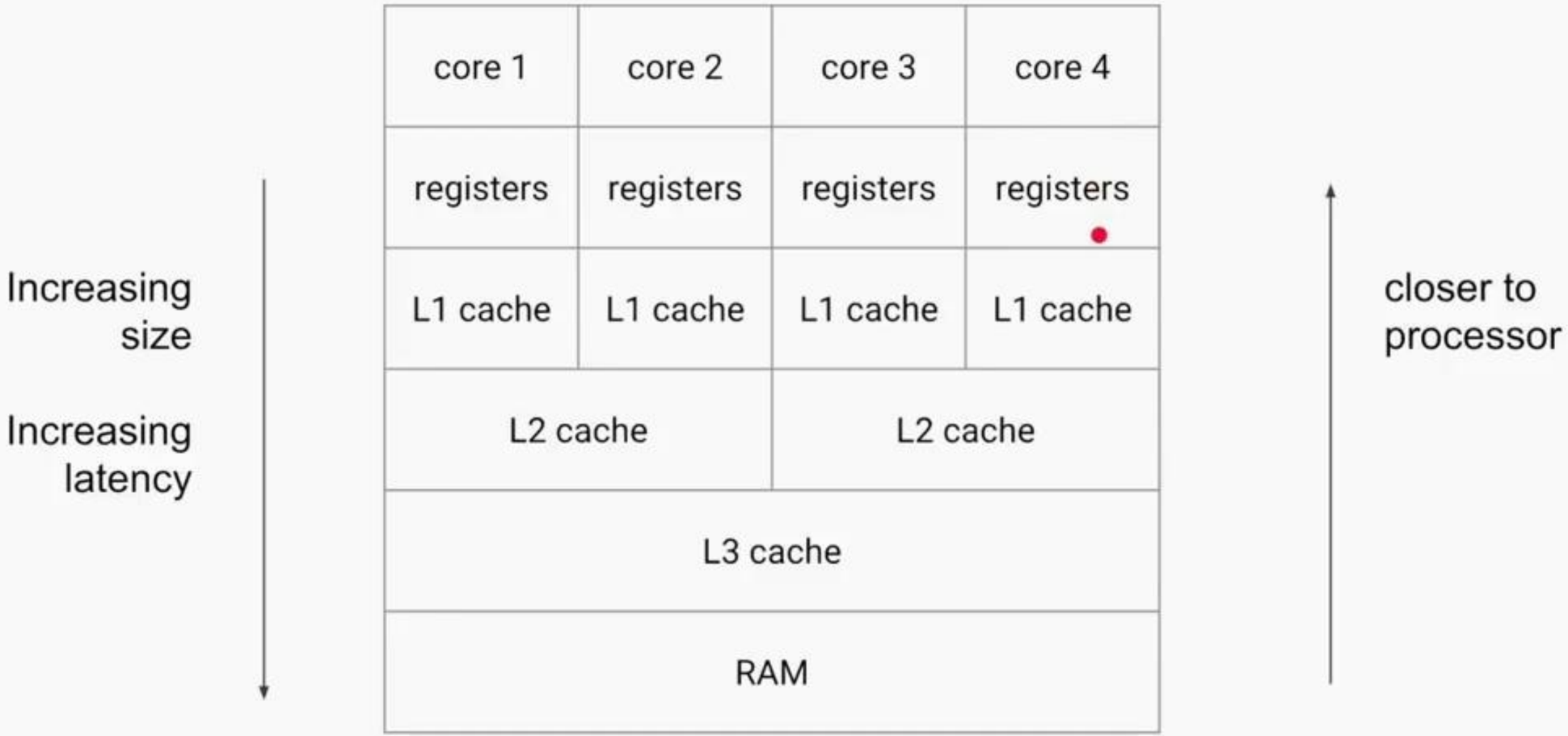
Reordered instructions

Field Visibility

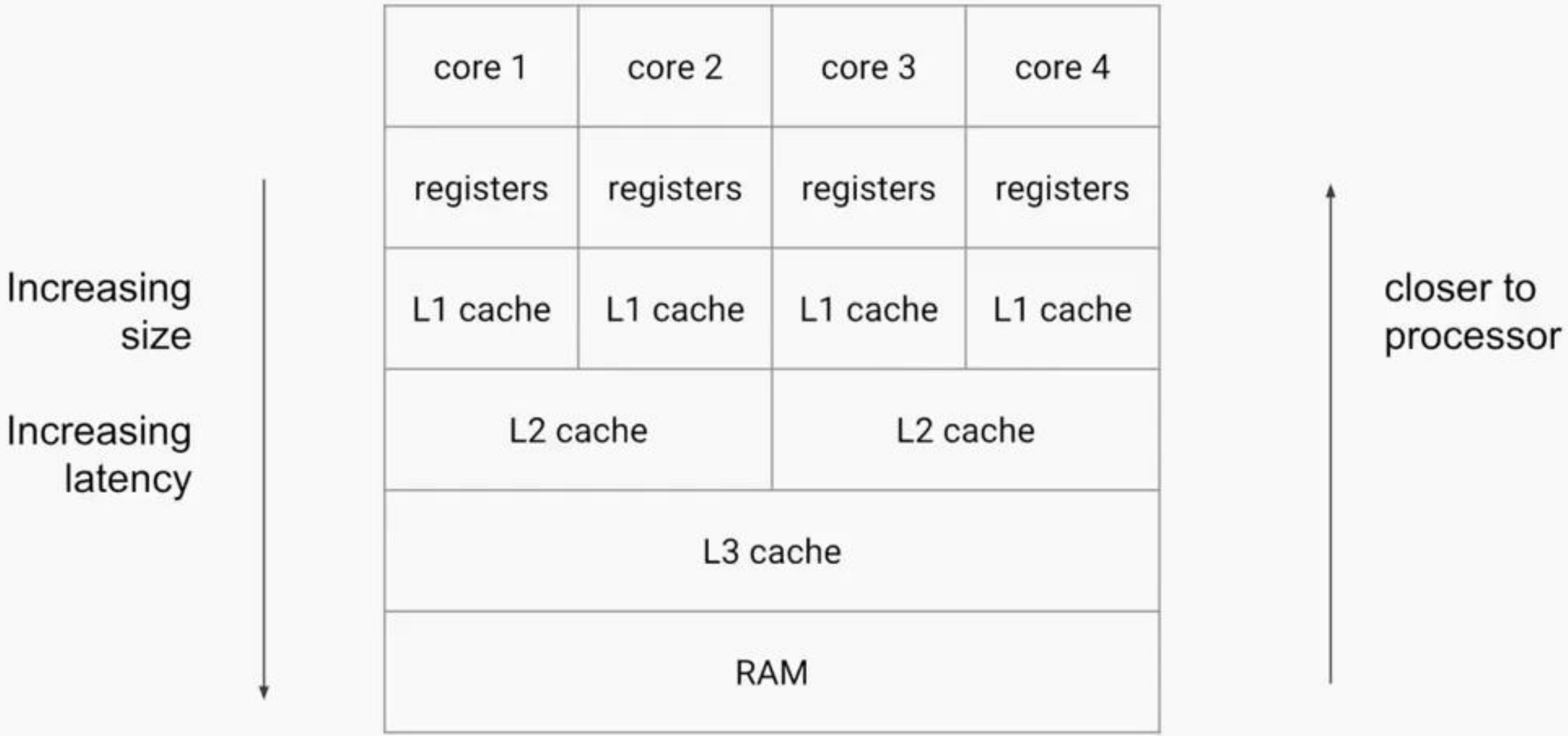
In presence of multiple threads

a.k.a

Concurrency



CPU caches (multi-core processor)



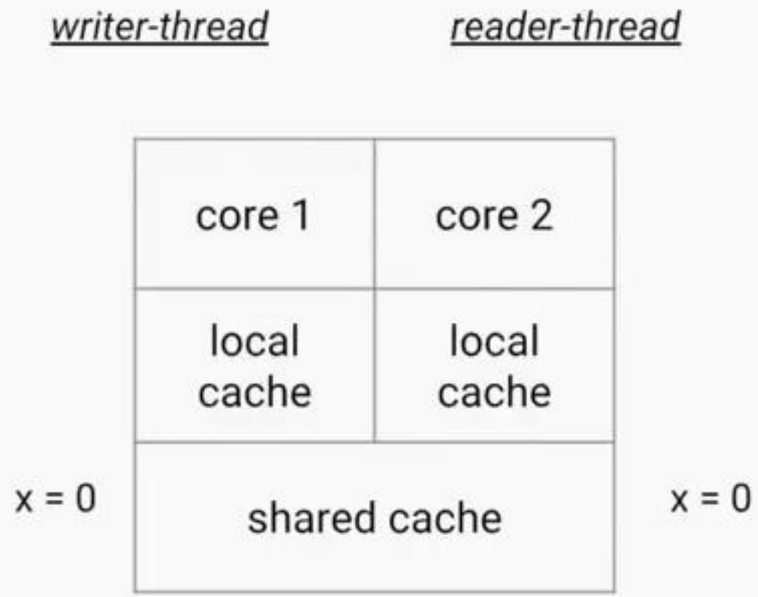
CPU caches (multi-core processor)

```java
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

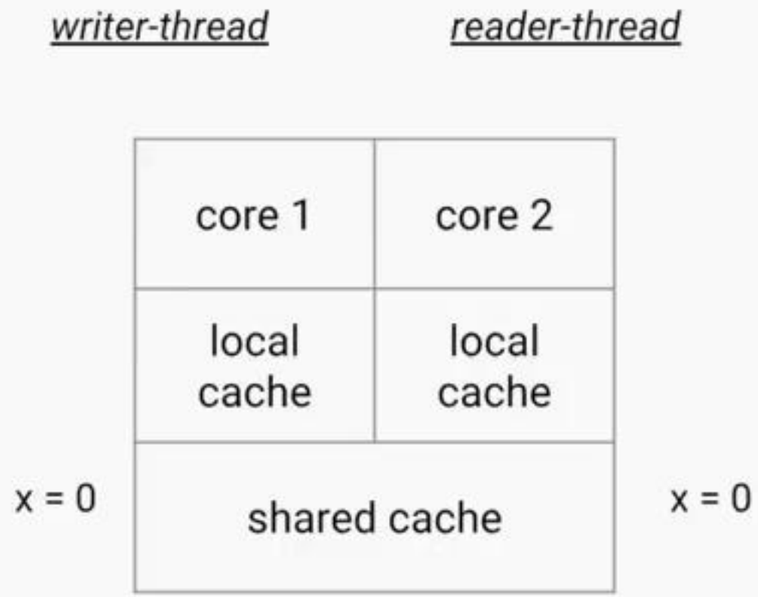


```java
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

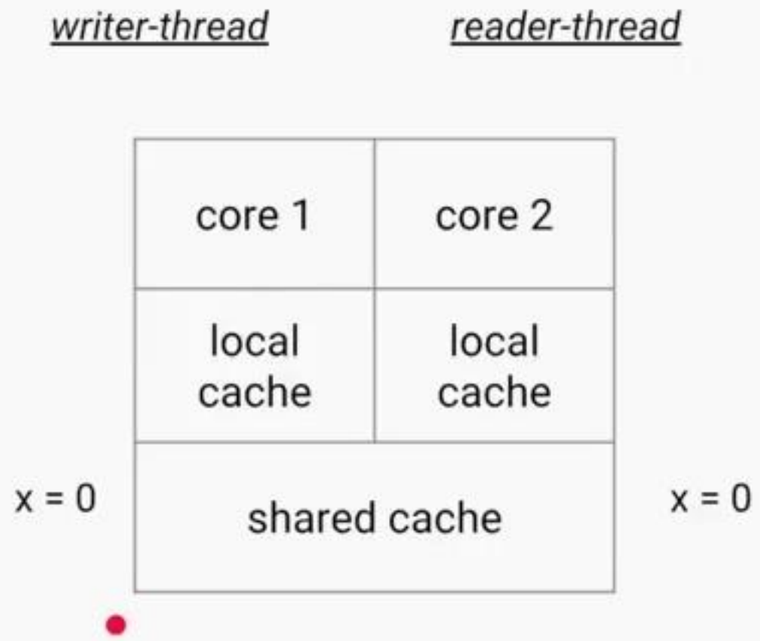


```java
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

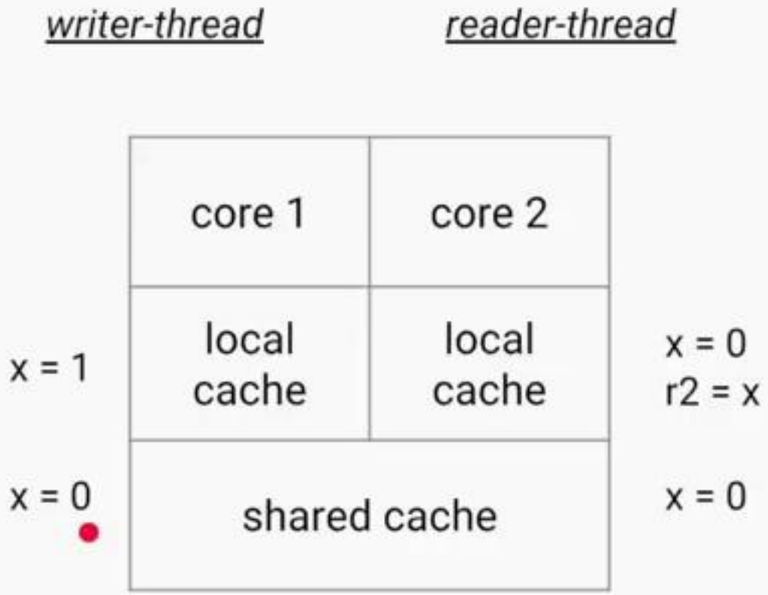


```java
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

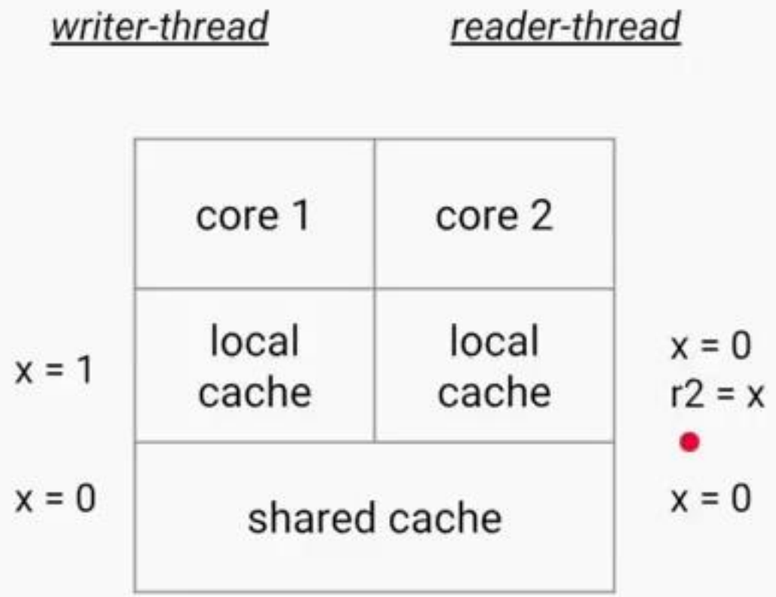


```java
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

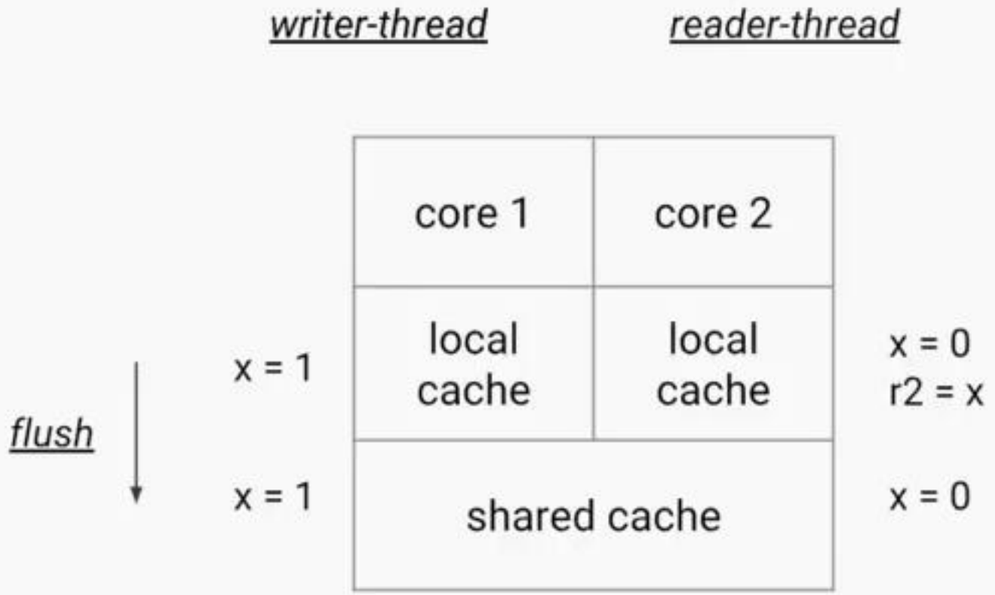


```java
public class VolatileVisibility {

    volatile int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```



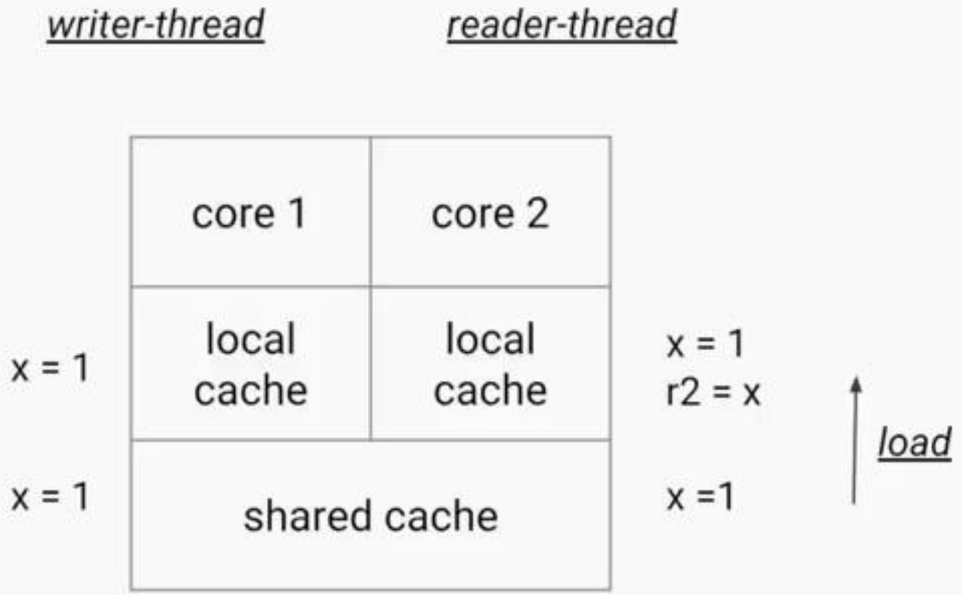
Field visibility sample with volatile

```java
public class VolatileVisibility {

    volatile int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```



Field visibility sample with volatile

What is Java Memory Model?

JMM is a specification which guarantees visibility of fields (aka happens before) amidst reordering of instructions.

What is Java Memory Model?

JMM is a specification which guarantees visibility of fields (aka happens before) amidst reordering of instructions.

```java
public class VolatileFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        a = 1;
        b = 1;
        c = 1;

        x = 1;  // write of x
    }

    public void readerThread() {

        int r2 = x;   // read of x

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

“Happens-before” relationship for volatile

```java
public class VolatileFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        a = 1;
        b = 1;
        c = 1;

        x = 1;  // write of x
    }

    public void readerThread() {

        int r2 = x;   // read of x

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

```
public class VolatileFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        a = 1;
        b = 1;
        c = 1;

        x = 1;  // write of x
    }

    public void readerThread() {

        int r2 = x;   // read of x

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

"Happens-before" relationship for volatile

Not only volatile

Also

1. Synchronized
2. Locks
3. Concurrent collections
4. Thread operations (join,start)

final fields (special behavior)

```java
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {
        a = 1;
        b = 1;
        c = 1;

        synchronized (this) {
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
        }

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

Same behavior with synchronized blocks. V.Imp: Has to be synchronized on same object!!

```java
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {
        a = 1;
        b = 1;
        c = 1;

        synchronized (this) {
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
        }

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

Same behavior with synchronized blocks. V.Imp: Has to be synchronized on same object!!

```java
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {
        a = 1;
        b = 1;
        c = 1;

        synchronized (this) {
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
        }

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

Sar nized on same object!!

```java
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        synchronized (this) {
            a = 1;
            b = 1;
            c = 1;
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
            int d1 = a;
            int d2 = b;
            int d3 = c;
        }
    }
}
```

Better to be more clear though!

```java
public class LockVisibility {

    int a = 0, b = 0, c = 0, x = 0;
    Lock lock = new ReentrantLock();

    public void writerThread() {

        lock.lock();
        a = 1;
        b = 1;
        c = 1;
        x = 1;
        lock.unlock();
    }

    public void readerThread() {

        lock.lock();
        int r2 = x;
        int d1 = a;
        int d2 = b;
        int d3 = c;
        lock.unlock();
    }
}
```

Same behavior with Locks.

```java
public class LockVisibility {

    int a = 0, b = 0, c = 0, x = 0;
    Lock lock = new ReentrantLock();

    public void writerThread() {

        lock.lock();
        a = 1;
        b = 1;
        c = 1;
        x = 1;
        lock.unlock();
    }

    public void readerThread() {

        lock.lock();
        int r2 = x;
        int d1 = a;
        int d2 = b;
        int d3 = c;
        lock.unlock();
    }
}
```

```java
public class VolatileVisibility {

    boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

wrong

```java
public class VolatileVisibility {

    volatile boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

right

Code on the left can have reader thread keep running. Fixed with volatile flag.

```java
public class VolatileVisibility {

    boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

wrong

```java
public class VolatileVisibility {

    volatile boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

right

Code on the left can have reader thread keep running. Fixed with volatile flag.

```java
public class VolatileVisibility {

    boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

wrong

```java
public class VolatileVisibility {

    volatile boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

right

Code on the left can have reader thread keep running. Fixed with volatile flag.

```java
public class VolatileVisibility {

    boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

wrong

```java
public class VolatileVisibility {

    volatile boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

right

Code on the left can have reader thread keep running. Fixed with volatile flag.

CONVERT YOUTUBE VIDEO INTO JPG/PDF IMAGES

WWW.GALLERYMKER.COM