Android Concurrency & Synchronization: Part 8

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

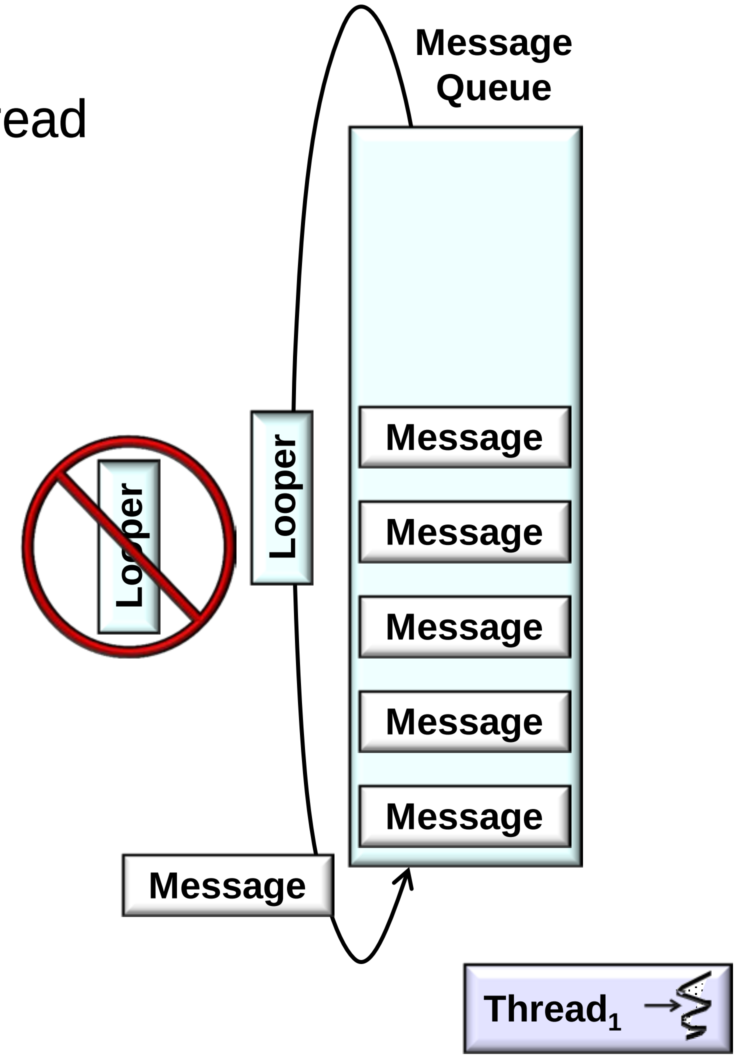
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II
Systems Programming for Android

Challenge: Ensuring One Looper per Thread

Context

- Android only allows one Looper per Thread



Challenge: Ensuring One Looper per Thread

Problem

- Using a central “Looper registry” in a multi-threaded process could become a bottleneck

Synchronization is required to serialize access by multiple threads

```
public class Looper {
  ...
  static final HashMap<long,
    Looper> looperRegistry = new
        HashMap<long, Looper>();
  ...
  private static void prepare() {
    synchronized(Looper.class) {
      Looper l = looperRegistry.
        get(Thread.getId());
      if (l != null)
        throw new
          RuntimeException("Only
            one Looper may be
            created per thread");
      looperRegistry.put
                (Thread.getId(),
                 new Looper());
    }
  ...
```

Challenge: Ensuring One Looper per Thread

Solution

- Apply the *Thread-Specific Storage* pattern to allow multiple threads to use one 'logically global' access point to retrieve the one & only Looper that is local to a thread

```
public class Looper {
  ...
  static final ThreadLocal
    <Looper> sThreadLocal = new
        ThreadLocal<Looper>();
  ...
  private static void prepare() {
    if (sThreadLocal.get()
        != null)
      throw new
        RuntimeException("Only
          one Looper may be
          created per thread");

     sThreadLocal.set(new
       Looper(quitAllowed));
  }
...
```

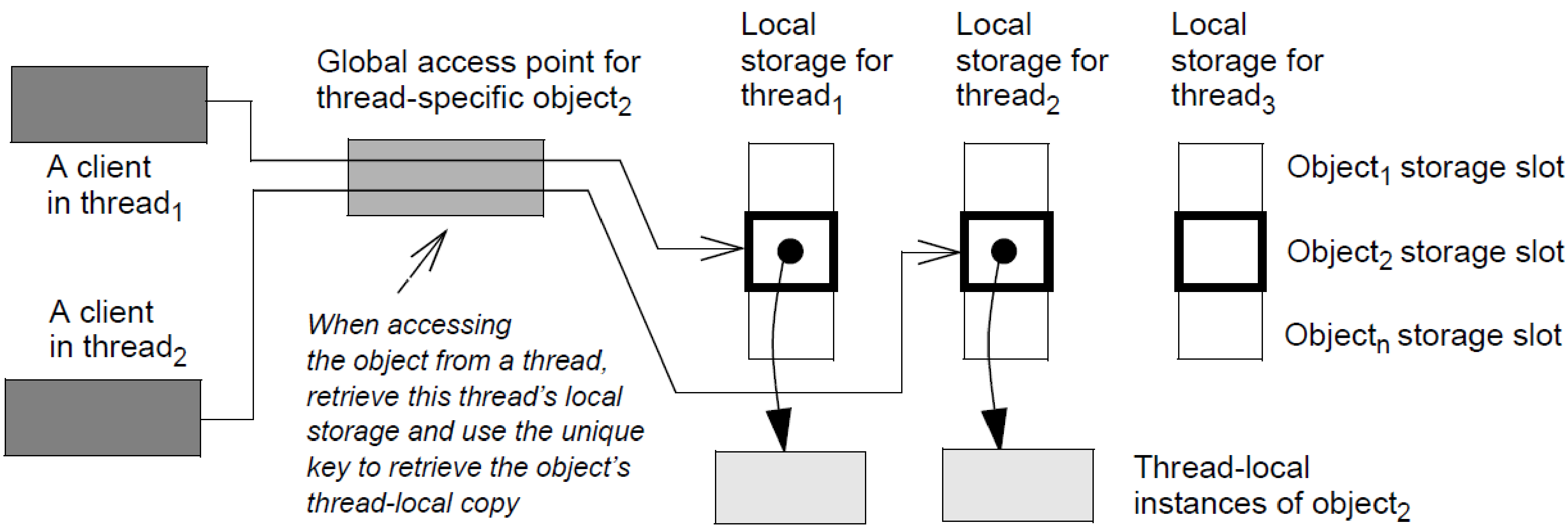
Thread-Specific Storage doesn't incur locking overhead on each object access

developer.android.com/reference/java/lang/ThreadLocal.html has more info

Thread-Specific Storage POSA2 Synchronization

Intent

- Allows multiple threads to use one ‘logically global’ access point to retrieve an object that is local to a thread, without incurring locking overhead on each object access



www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf has more info

Thread-Specific Storage POSA2 Synchronization

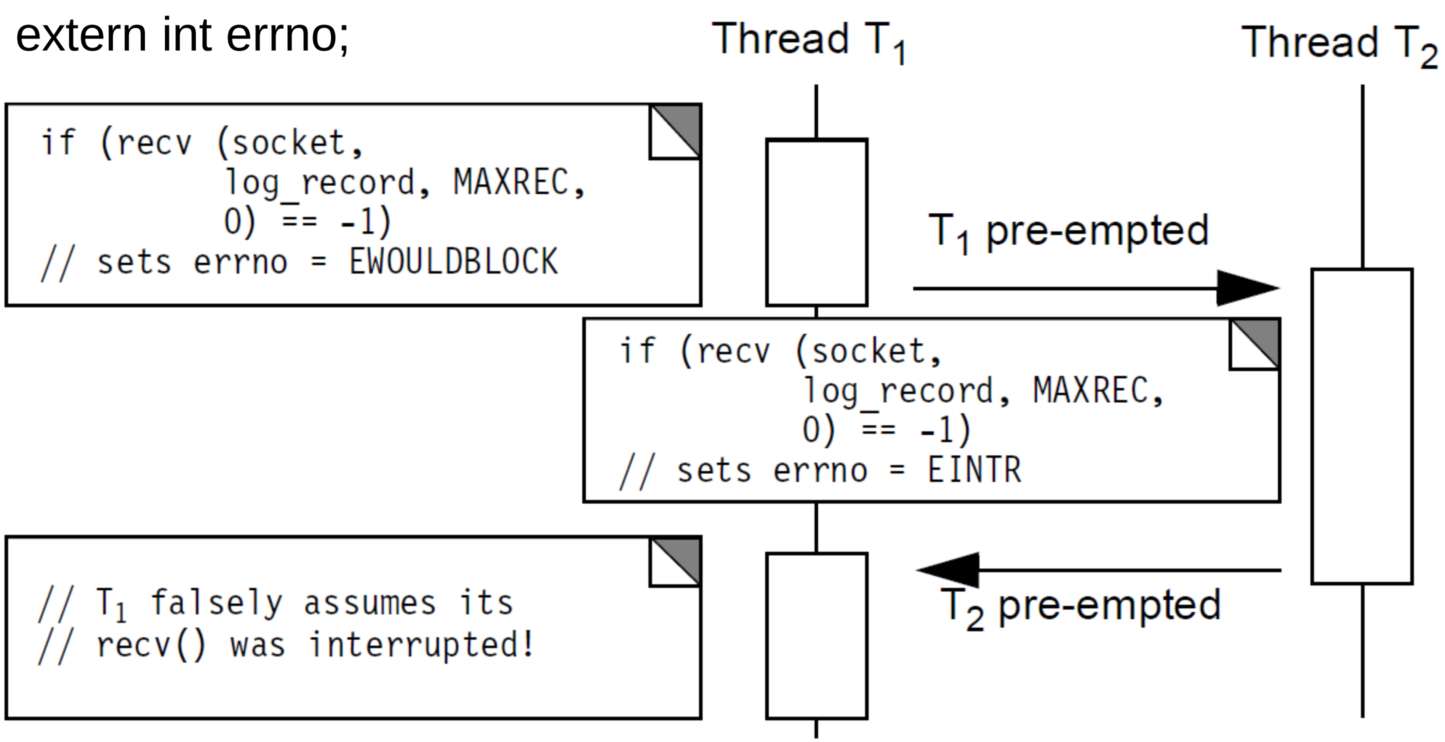
Applicability

- You want a concurrent program that is both easy to program & efficient
  - e.g., access to data that is logically global—but physically local to a thread—should be *efficiently* atomic

Thread-Specific Storage POSA2 Synchronization

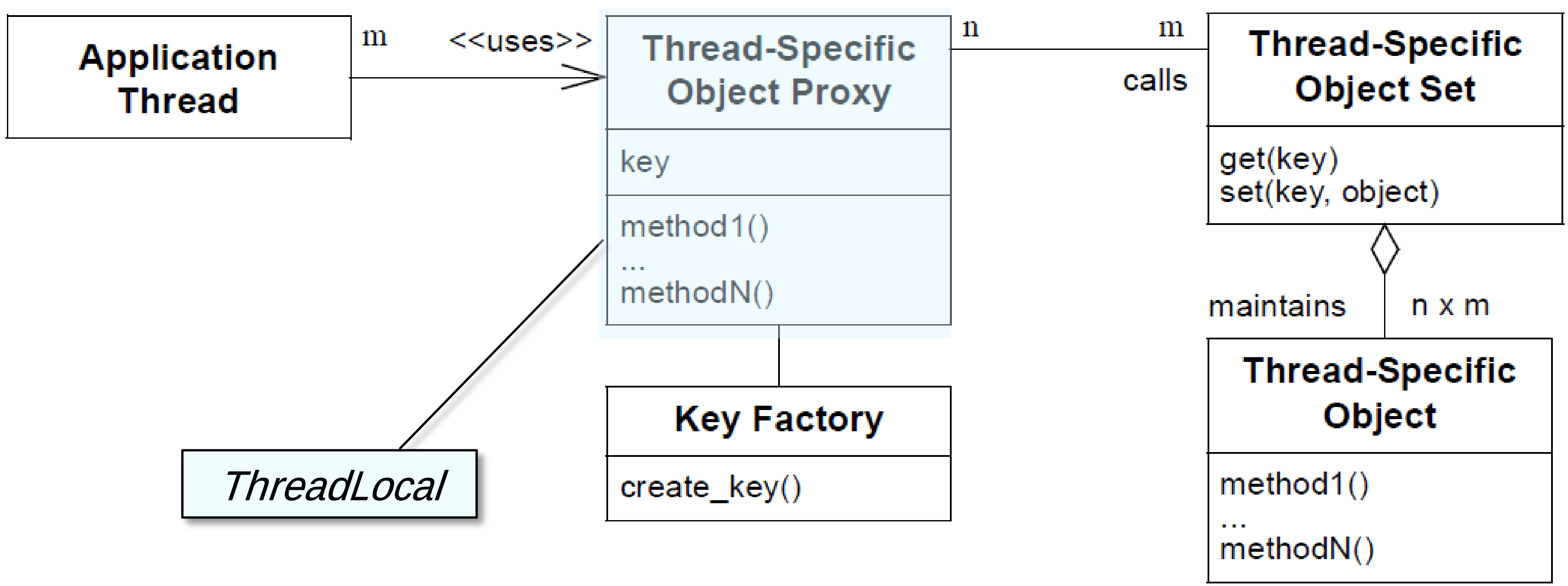
Applicability

- You want a concurrent program that is both easy to program & efficient
- You need to retrofit legacy code to be thread-safe
  - Many legacy libraries & apps written assuming a single thread of control pass data implicitly between methods via global objects



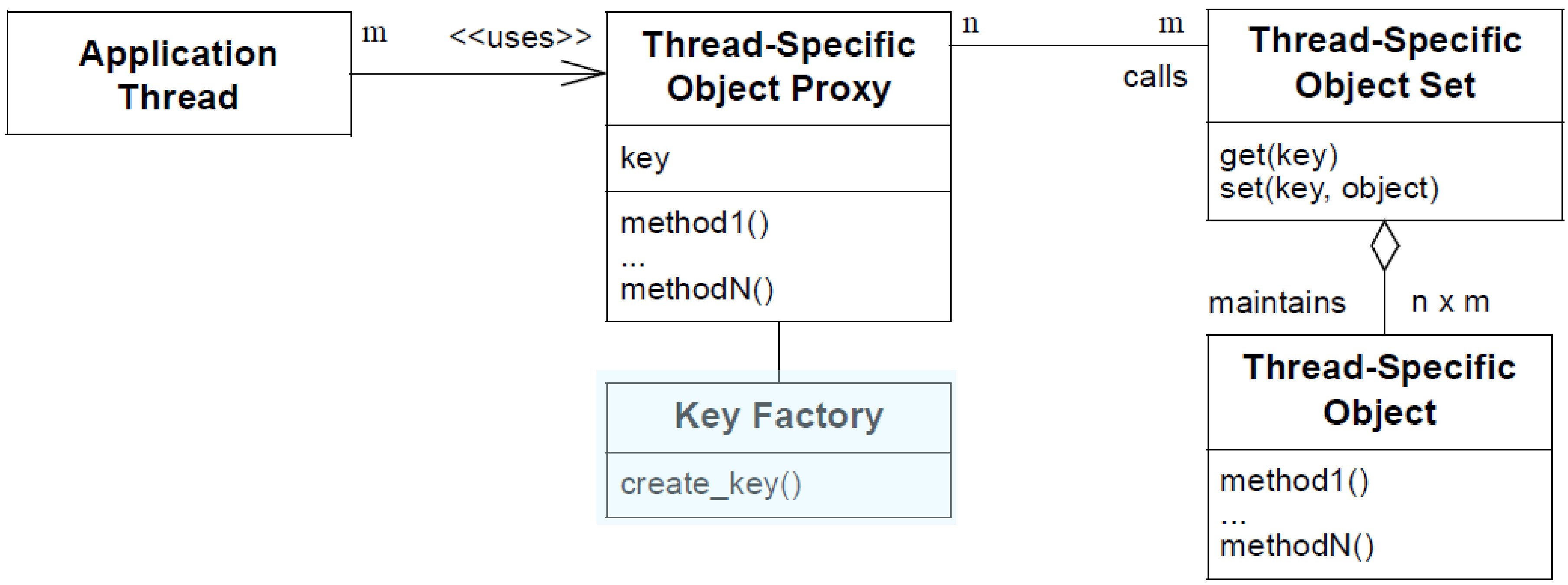
Thread-Specific Storage POSA2 Synchronization

Structure & Participants



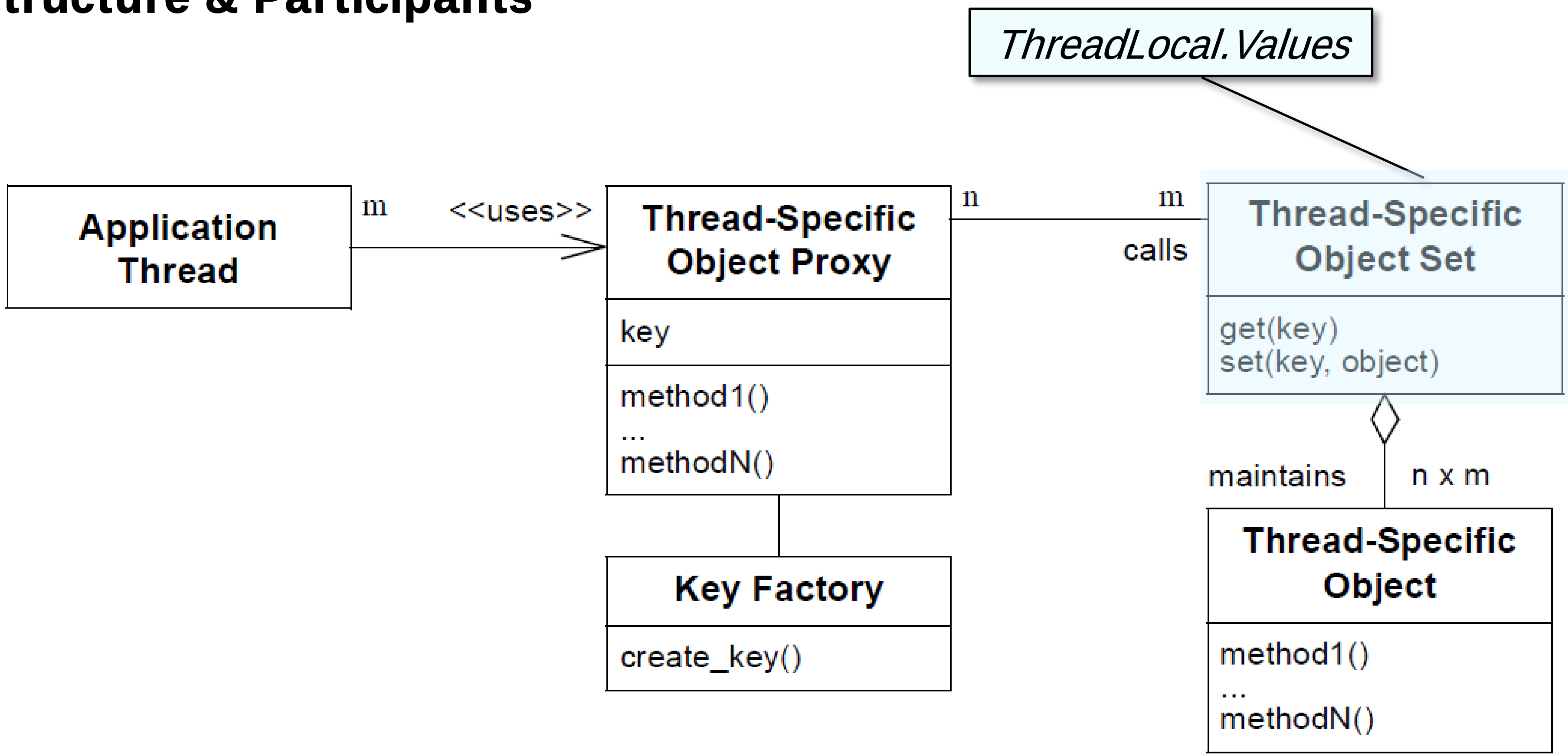
Thread-Specific Storage POSA2 Synchronization

Structure & Participants



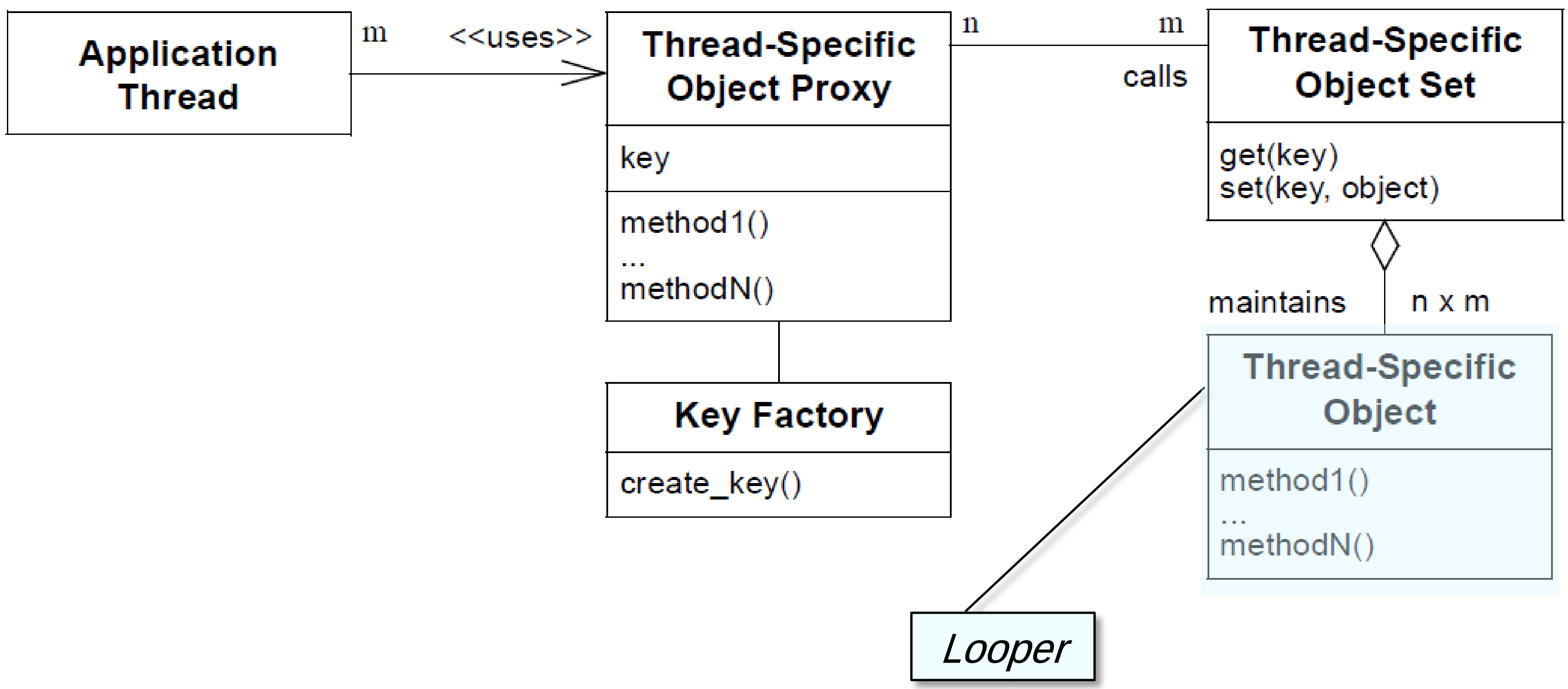
Thread-Specific Storage POSA2 Synchronization

Structure & Participants



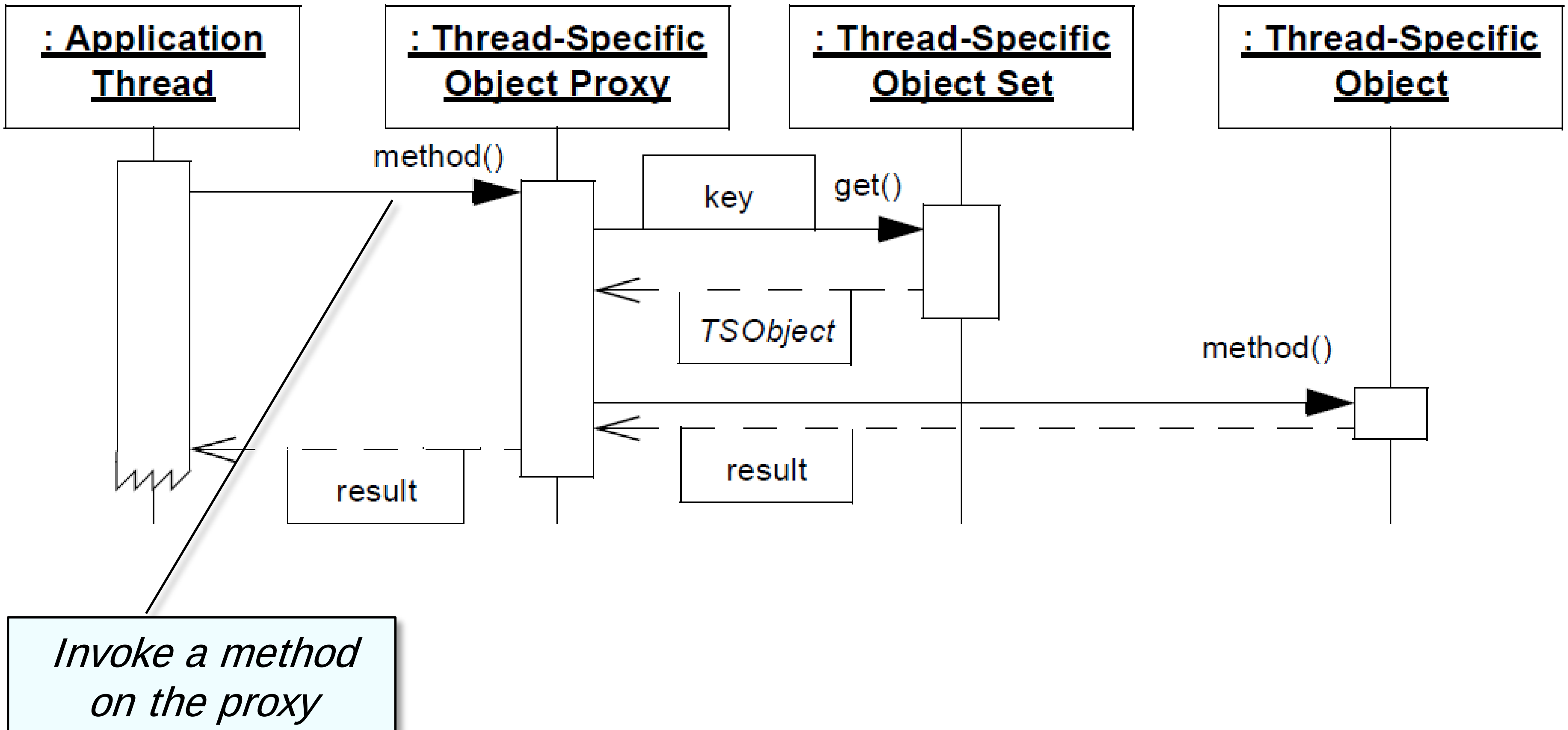
Thread-Specific Storage POSA2 Synchronization

Structure & Participants



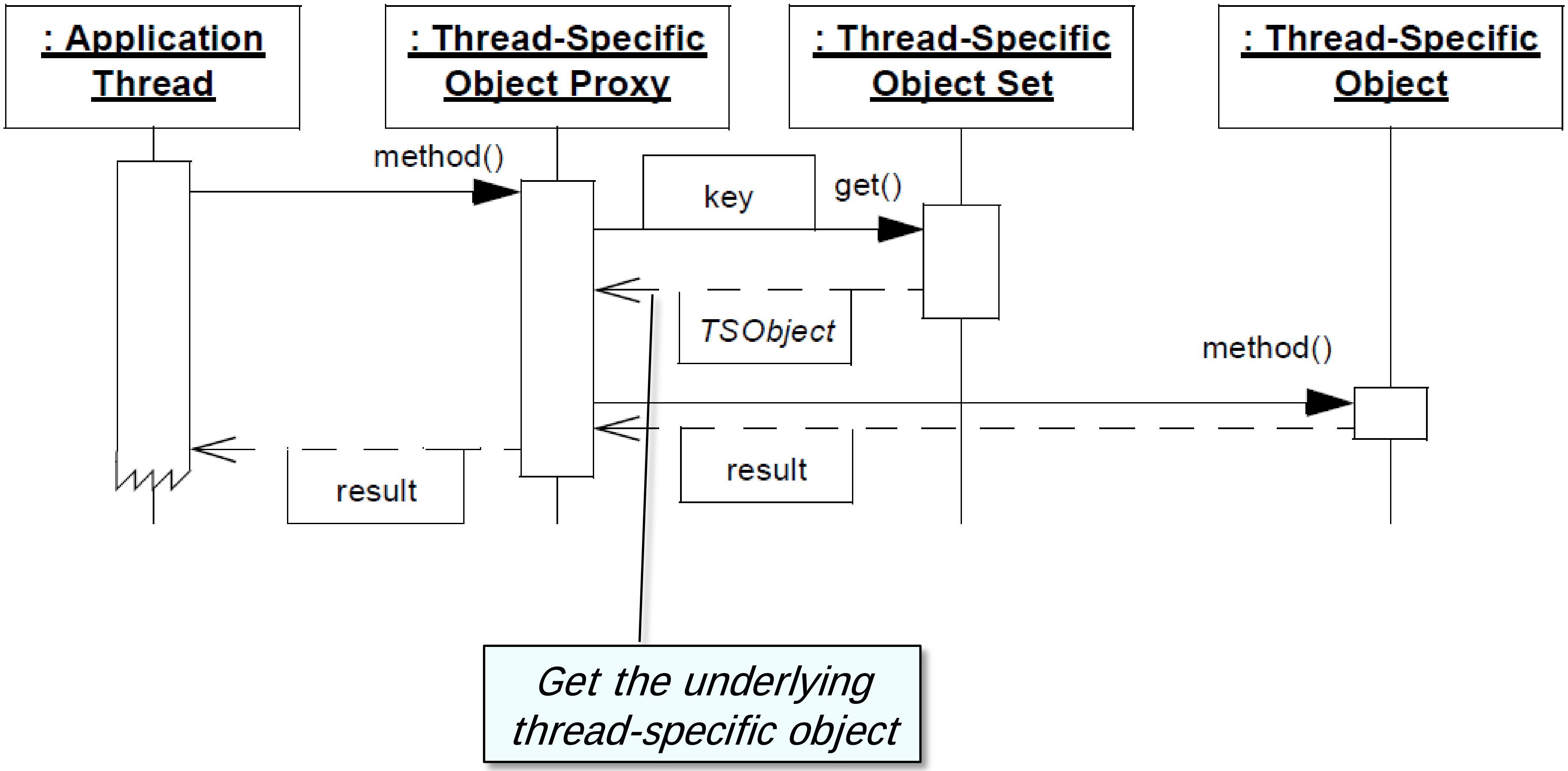
Thread-Specific Storage POSA2 Synchronization

Dynamics



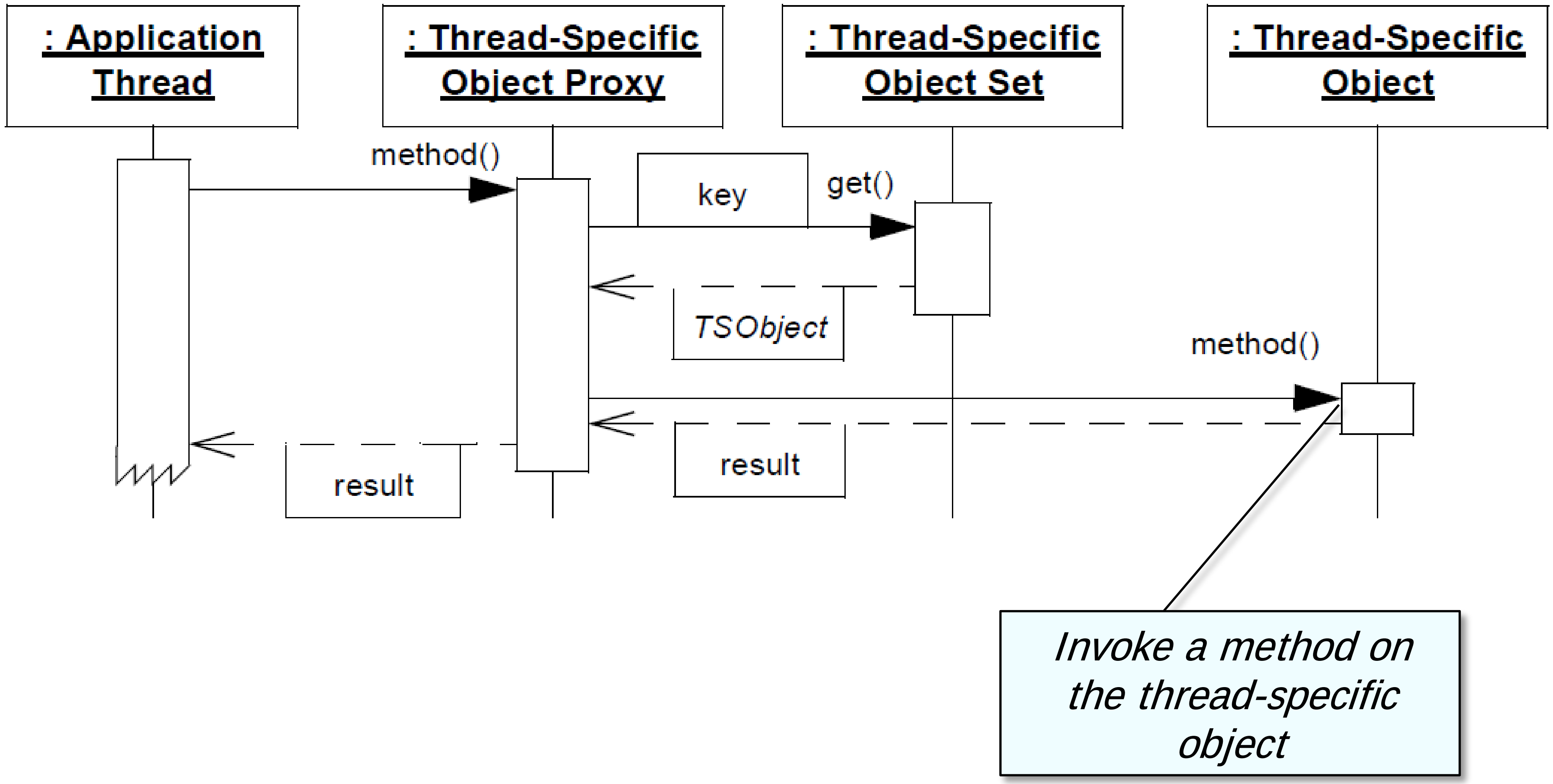
Thread-Specific Storage POSA2 Synchronization

Dynamics



Thread-Specific Storage POSA2 Synchronization

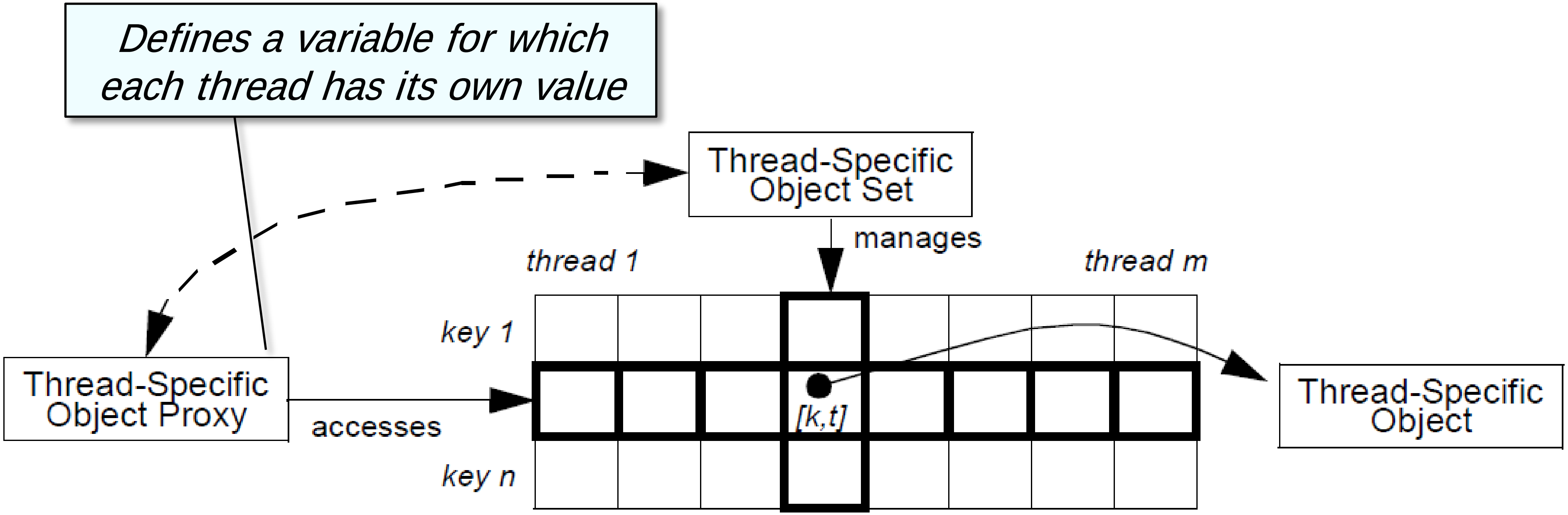
Dynamics



Thread-Specific Storage POSA2 Synchronization

Implementation

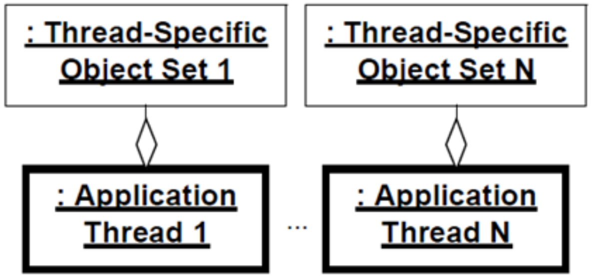
- Implement thread-specific object proxies
  - Mediates access to the underlying thread-specific objects



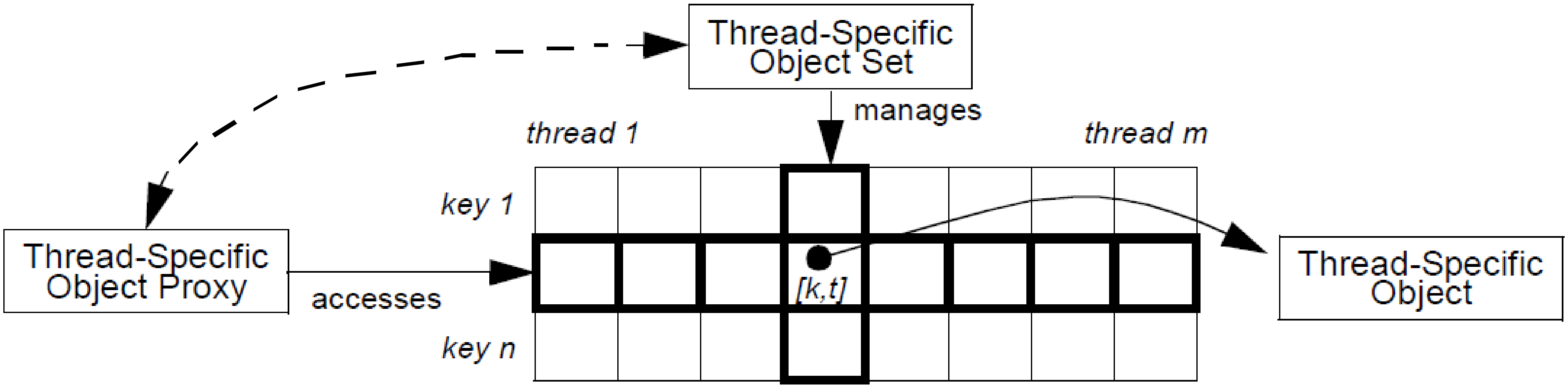
Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives:



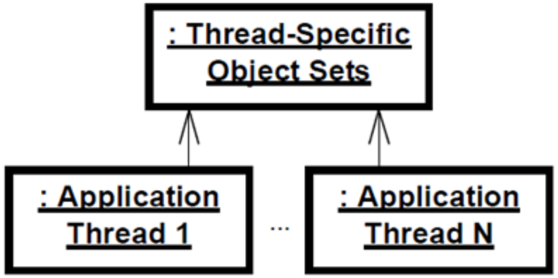
Thread-internal Thread-Specific Object Set



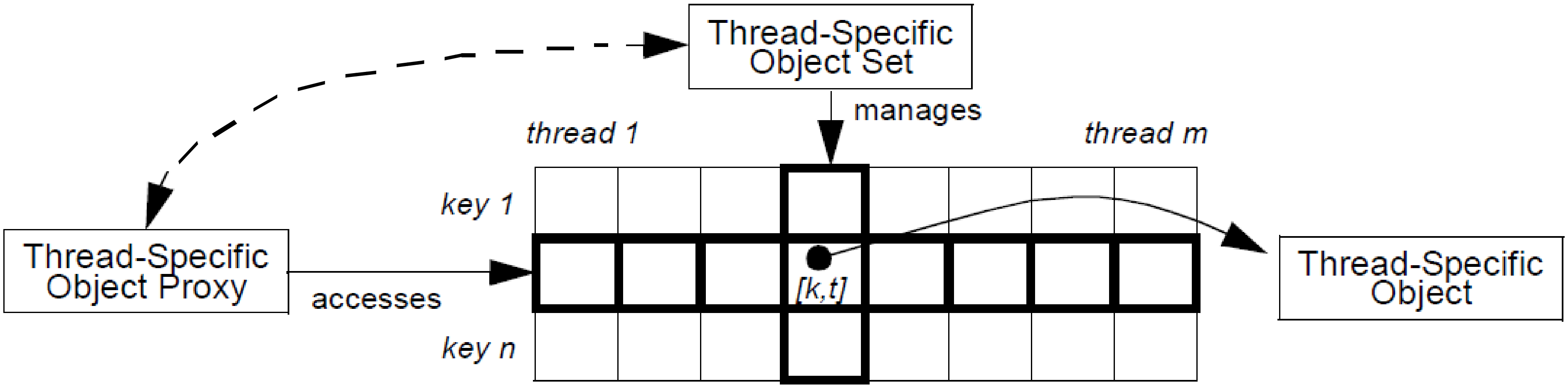
Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives:



Thread-external Thread-Specific Object Set

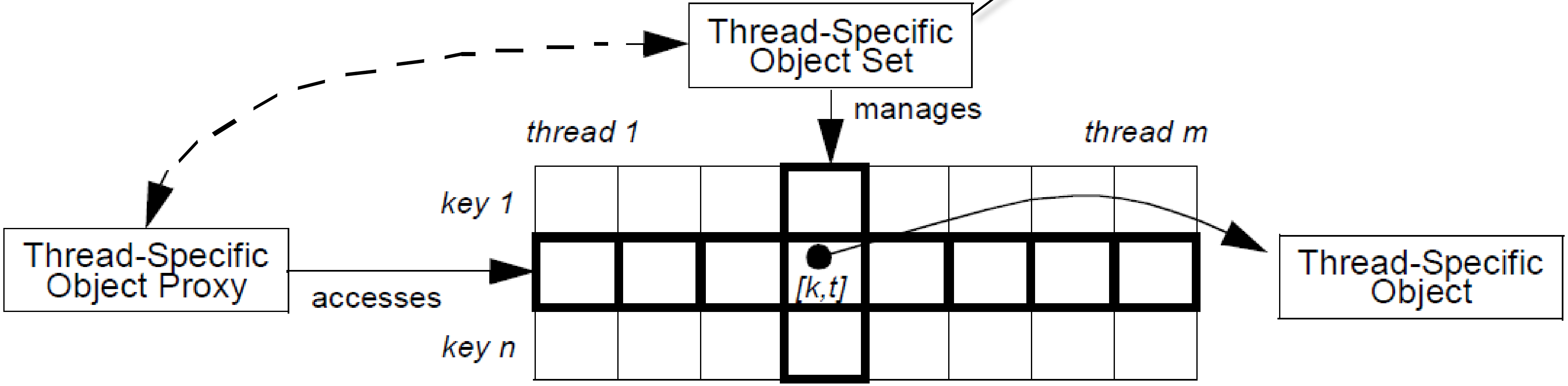


Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - Define data structures that map keys & thread ids to thread-specific object sets

Per-thread data structure that maps keys to thread-specific objects

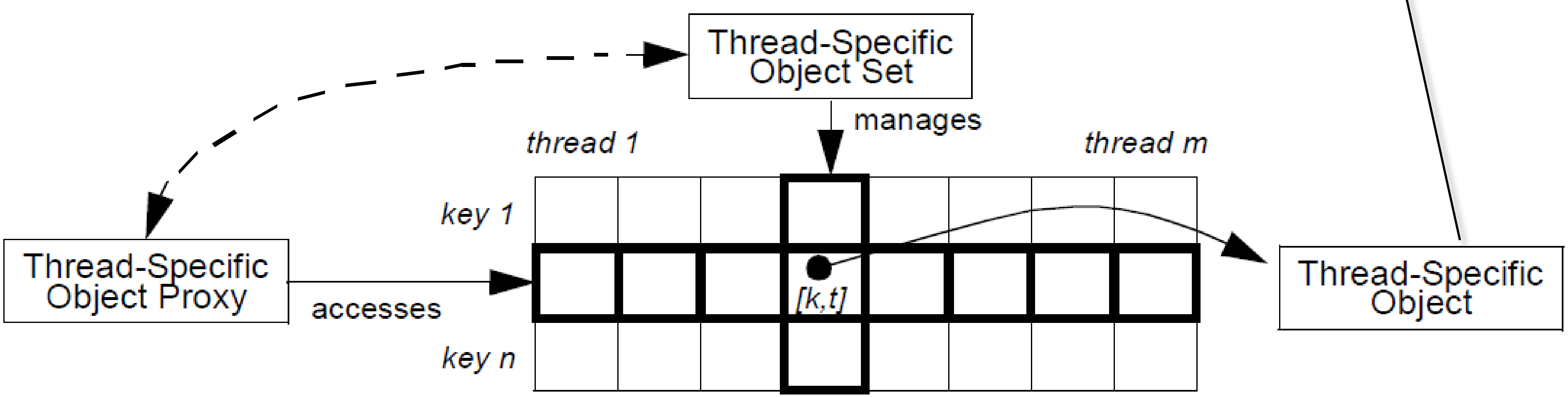


Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - Define data structures that map keys & thread ids to thread-specific object sets

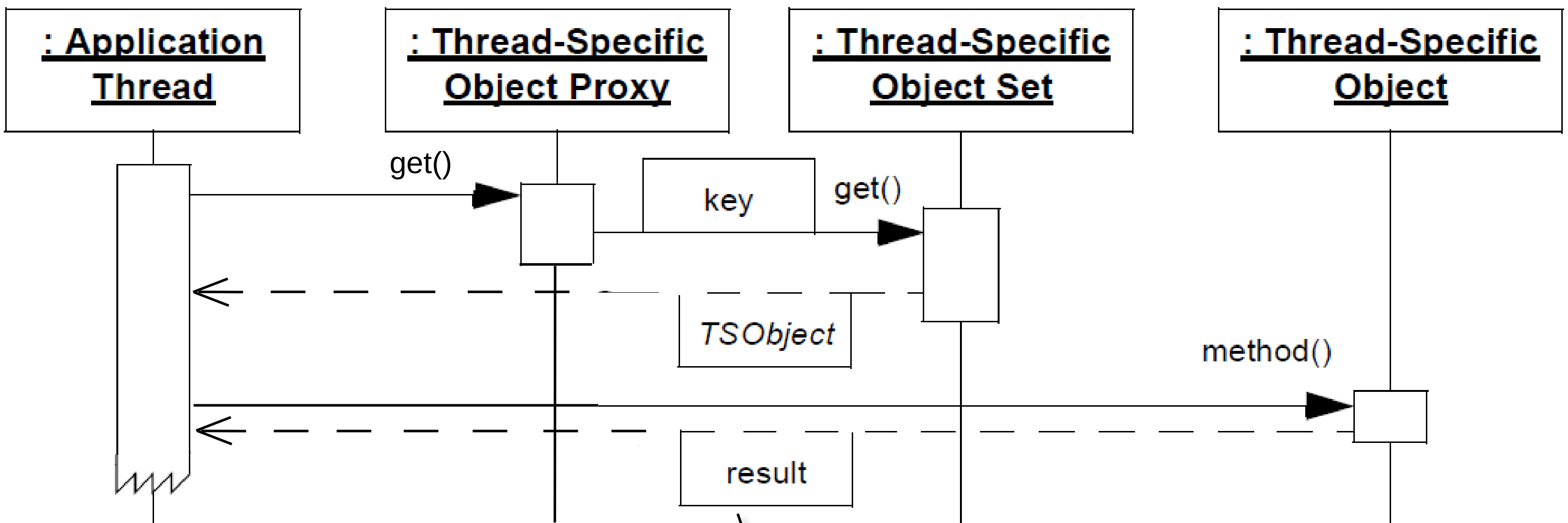
The thread identifier, thread-specific object set, & the proxy cooperate to obtain the correct thread-specific object



Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern



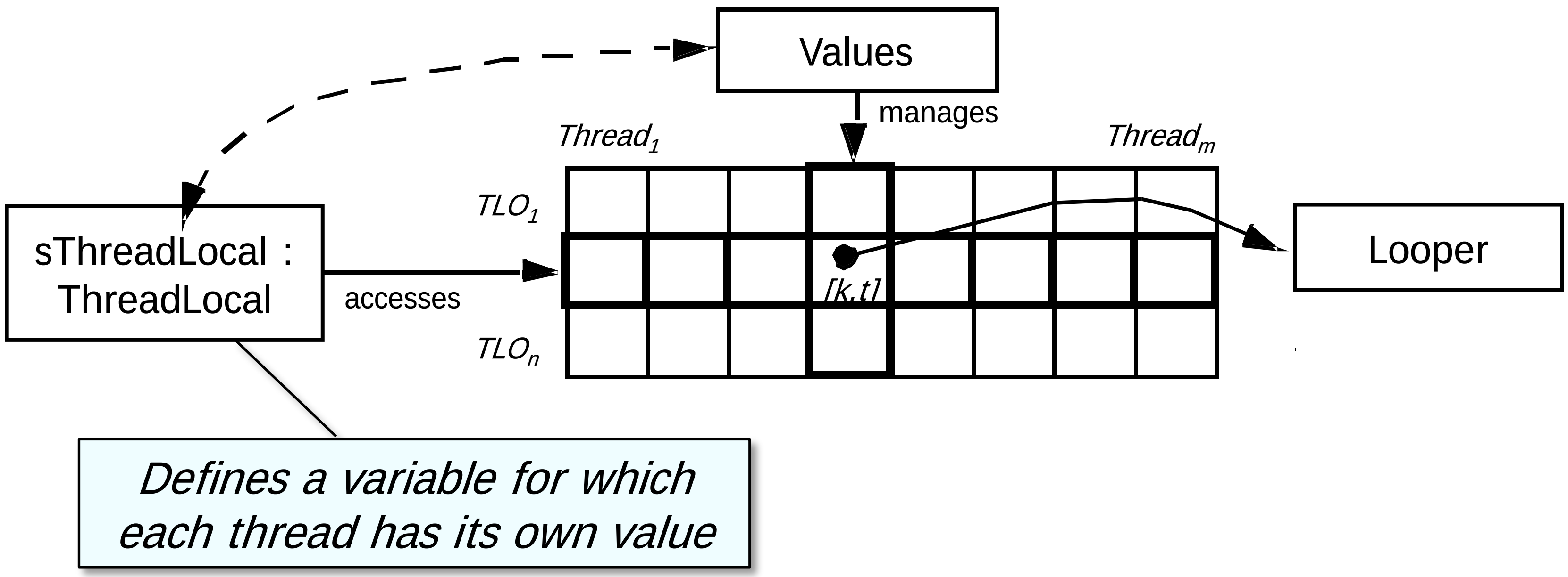
The Java ThreadLocal class doesn't use a full-blown proxy

developer.android.com/reference/java/lang/ThreadLocal.html has more info

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

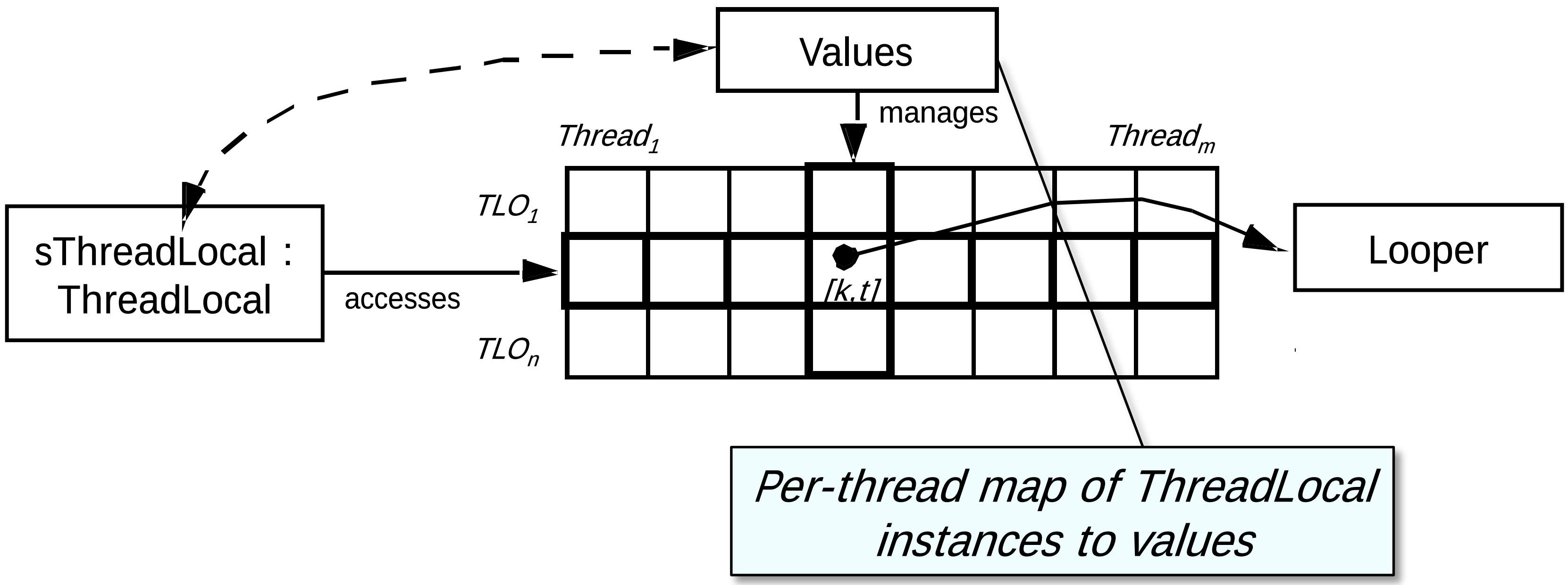


- All threads share the same ThreadLocal object

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

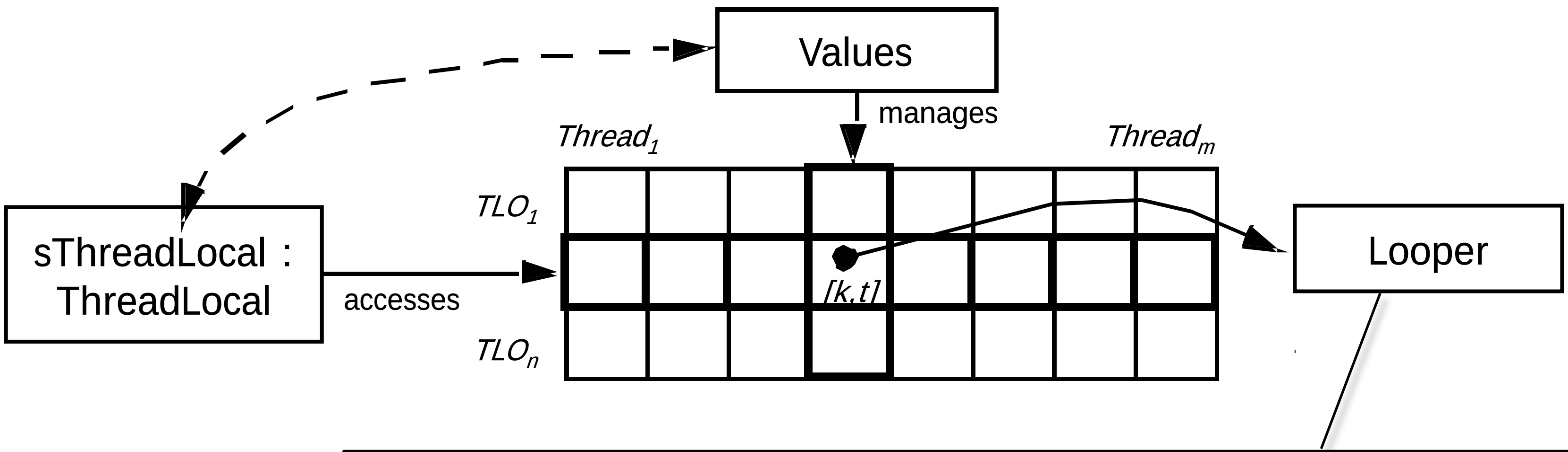


- All threads share the same ThreadLocal object
- Each thread sees a different value when accessing it, so changes made by one thread don't affect values of ThreadLocal objects in other threads

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern



The thread identifier, Values map, & the ThreadLocal object cooperate to obtain the correct thread-specific Looper object

- All threads share the same ThreadLocal object
- Each thread sees a different value when accessing it, so changes made by one thread don't affect values of ThreadLocal objects in other threads

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- The ThreadLocal.set() method identifies the corresponding Values map based on the current Thread Id & *stores* the value

```
public class ThreadLocal<T> {
  ...
  public void set(T value) {
    Thread currentThread =
      Thread.currentThread();
    Values values =
      values(currentThread);
    if (values == null) {
      values =
        initializeValues
          (currentThread);
    }
    values.put(this, value);
  }
  ...
```

Note there's no synchronization involved at all!

libcore/luni/src/main/java/java/lang/ThreadLocal.java has the source code

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- The ThreadLocal.set() method identifies the corresponding Values map based on the current Thread Id & *stores* the value
- The ThreadLocal.get() method does the same thing, but *returns* the thread-specific object

```
public class ThreadLocal<T> {
  ...
  public T get() {
    Thread currentThread =
      Thread.currentThread();
    Values values =
      values(currentThread);
    if (values != null) {
      Object[] table =
        values.table;
      int index = hash &
        values.mask;
      if (this.reference ==
          table[index]) {
        return (T)
          table[index + 1];
  ...
```

Note there's no synchronization involved at all!

libcore/luni/src/main/java/java/lang/ThreadLocal.java has the source code

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread

```
public class Looper {
  ...
  static final
    ThreadLocal<Looper>
      sThreadLocal = new
        ThreadLocal<Looper>();
  ...
  private static void prepare() {
    if (sThreadLocal.get()
        != null)
      throw new
        RuntimeException("Only
          one Looper may be
          created per thread");
    sThreadLocal.set(new
      Looper(quitAllowed));
   ...
```

Thread-Specific Storage doesn't incur locking overhead on each object access

developer.android.com/reference/java/lang/ThreadLocal.html has more info

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static Looper myLooper()
  { return sThreadLocal.get(); }

  public static void loop() {
    final Looper me = myLooper();
    if (me == null)
      throw new RuntimeException
        ("No Looper; Looper.
          prepare() wasn't called
          on this thread.");
    final MessageQueue queue =
      me.mQueue;
  ...
```

Cache Looper instance data from the thread-specific Looper object

frameworks/base/core/java/android/os/Looper.java has the source code

Thread-Specific Storage POSA2 Synchronization

Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods
- The Handler constructor also uses myLooper() to connect a Handler the Thread where it's created

```
public class Handler {
  ...
  public Handler() {
    mLooper = Looper.myLooper();
    if (mLooper == null)
      throw new RuntimeException
        ("Can't create handler
         inside thread that has
         not called Looper.
         prepare()");

    mQueue = mLooper.mQueue;
    mCallback = null;
  }
  ...
```

frameworks/base/core/java/android/os/Handler.java has the source code

Thread-Specific Storage POSA2 Synchronization

Consequences

+ Efficiency
  - It's possible to implement this pattern so that no locking is needed to access thread-specific data

```
public class ThreadLocal<T> {
  ...
  public void set(T value) {
    Thread currentThread =
      Thread.currentThread();
    Values values =
      values(currentThread);
    if (values == null) {
      values =
        initializeValues
          (currentThread);
    }
    values.put(this, value);
  }
  ...
```

Thread-Specific Storage POSA2 Synchronization

Consequences

+ Efficiency

+ Ease of use

- When encapsulated with wrapper facades & proxies, thread-specific storage is easy for app developers to use

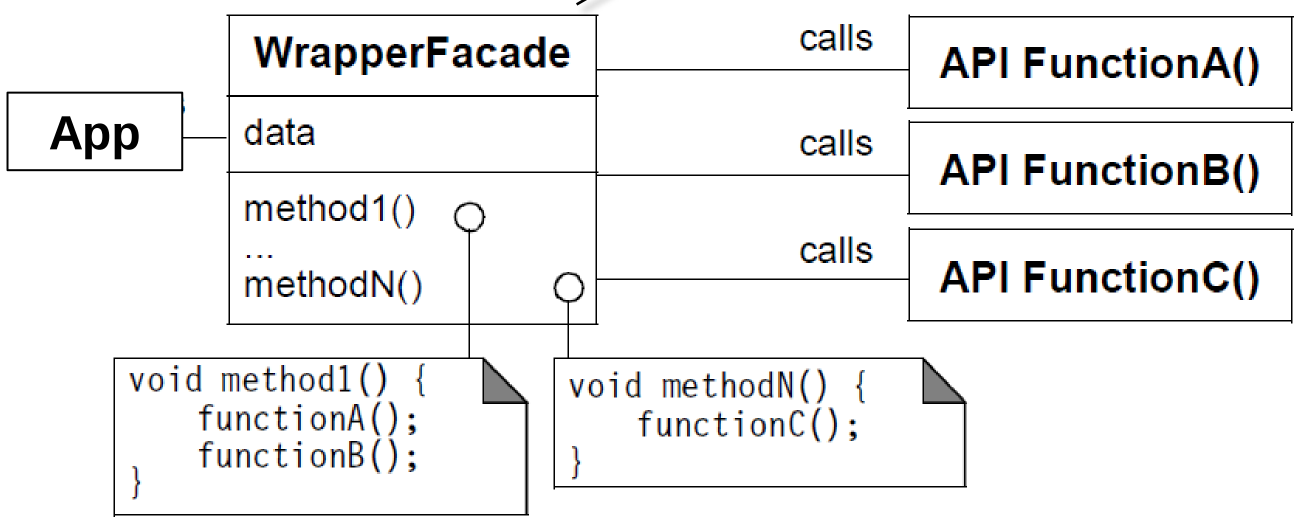
```
public class Looper {
  ...
  static final
    ThreadLocal<Looper>
      sThreadLocal = new
        ThreadLocal<Looper>();
  ...
  private static void prepare() {
    if (sThreadLocal.get()
          != null)
      throw new
        RuntimeException("Only
          one Looper may be
          created per thread");
   sThreadLocal.set(new
     Looper(quitAllowed));
   ...
```

Thread-Specific Storage POSA2 Synchronization

Consequences

+ Efficiency
+ Ease of use
+ Reusability & Portability
  - By combining this pattern with the *Wrapper Façade* pattern it's possible to shield developers from non-portable OS platform characteristics

Wrapper Façade encapsulates data & functions provided by existing C APIs within more concise, robust, portable, maintainable, & cohesive object-oriented classes



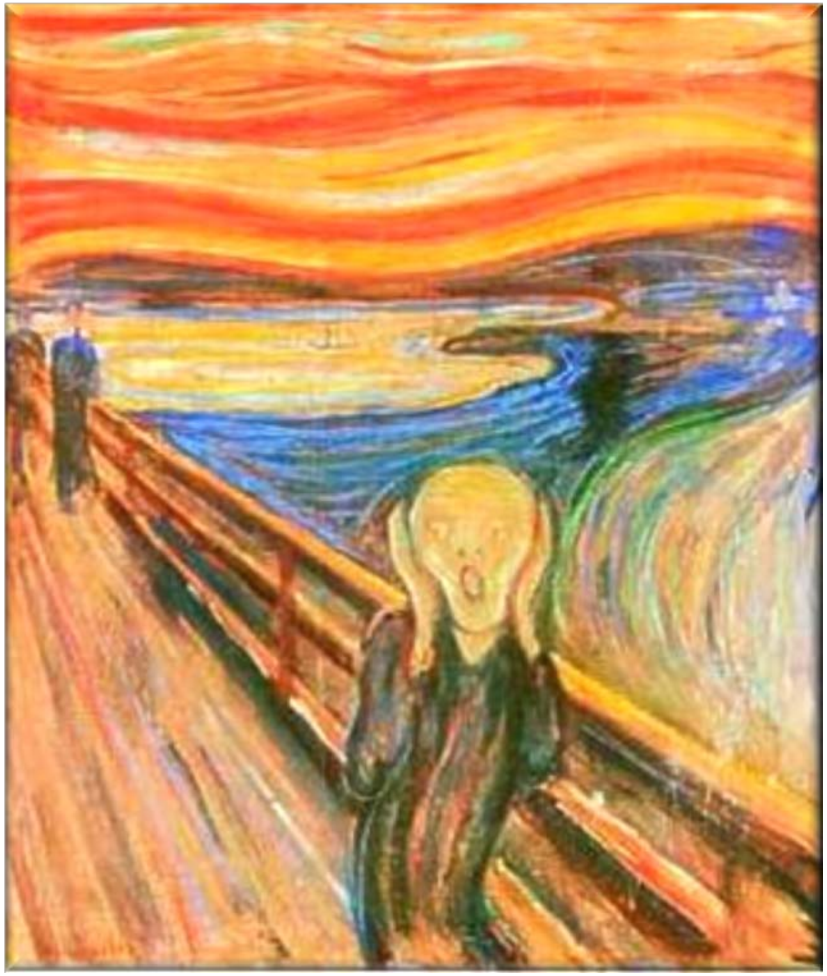
See www.dre.vanderbilt.edu/~schmidt/PDF/wrapper-facade.pdf for info

Thread-Specific Storage POSA2 Synchronization

Consequences

– It encourages use of thread-specific global objects

- Many apps do not require multiple threads to access thread-specific data via a common access point
  - In this case, data should be stored so that only the object owning the data can access it



Thread-Specific Storage POSA2 Synchronization

Consequences

– It encourages use of thread-specific global objects

– It obscures the structure of the system

- The use of thread-specific storage potentially makes an app hard to understand, by obscuring relationships between its components

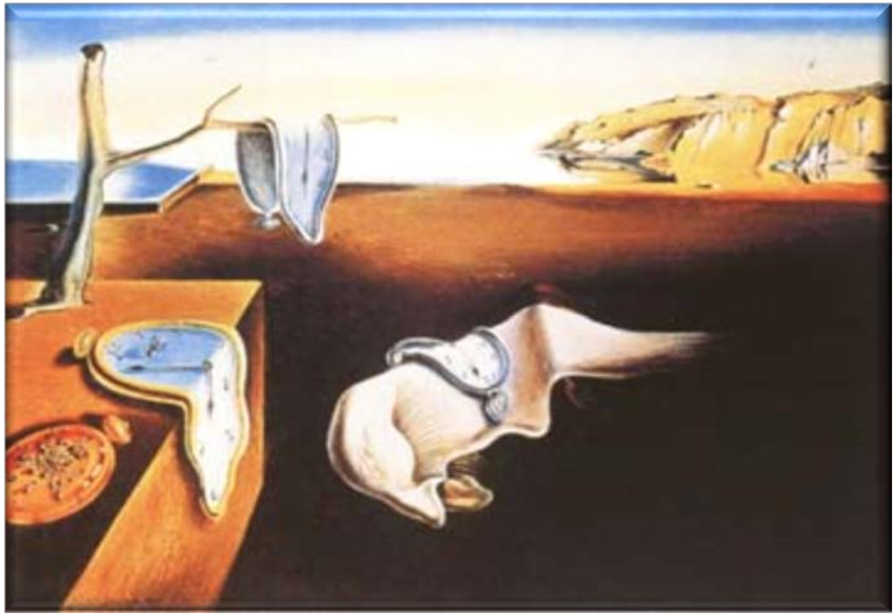
Thread-Specific Storage POSA2 Synchronization

Consequences

– It encourages use of thread-specific global objects

– It obscures the structure of the system

– Inefficient implementations may be slower than using a lock!



Thread-Specific Storage POSA2 Synchronization

Known Uses

- The errno macro in multi-threaded implementations of standard C

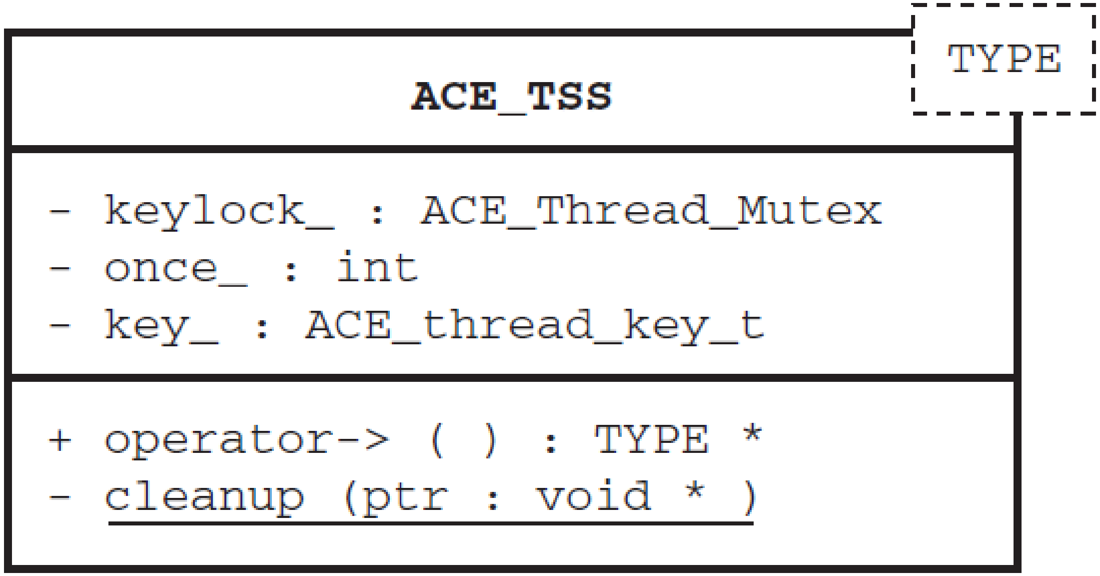
```
// ...
if ((n = recv (fd,
               buf,
               sizeof (buf)))
    == -1
    && errno != EWOULDBLOCK)
  ...
};
```

- *The errno macro expands to an lvalue with type int, containing the last error code generated in any function using the errno facility*
- *Originally this was a static memory location, but macros are almost always used today to allow for multi-threading, each thread will see its own error number*

Thread-Specific Storage POSA2 Synchronization

Known Uses

- The errno macro in multi-threaded implementations of standard C
- ACE_TSS template
  - Encapsulates & enhances native OS Thread-Specific Storage (TSS) APIs



www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf has more info

Thread-Specific Storage POSA2 Synchronization

Known Uses

- The errno macro in multi-threaded implementations of standard C
- ACE_TSS template
  - Encapsulates & enhances native OS Thread-Specific Storage (TSS) APIs
  - It uses C++ operator>() (delegation operator) to provide thread-specific smart pointers

```
template <class TYPE> TYPE *
ACE_TSS<TYPE>::operator-> () {
  ...
  TYPE *ts_obj = 0;

  ACE_OS::thr_getspecific (key_,
                (void **) &ts_obj);
  if (ts_obj == 0) {
    ts_obj = new TYPE;
```

Dynamically allocate the TYPE object if it doesn't already exist

```
    ACE_OS::thr_setspecific (key_,
                             ts_obj);
  }
  return ts_obj;
}
```

Thread-Specific Storage POSA2 Synchronization

Known Uses

- The errno macro in multi-threaded implementations of standard C
- ACE_TSS template
  - Encapsulates & enhances native OS Thread-Specific Storage (TSS) APIs
  - It uses C++ operator>() (delegation operator) to provide thread-specific smart pointers

```
class Request_Count {
public:
  Request_Count (): c_ (0) {}
  void increment () { ++c_; }
  int value () const { return c_; }
private:
  int c_;
};

ACE_TSS<Request_Count>
        request_count;

...

request_count->increment ();
```

This call increments the Request_Count object in thread-specific storage

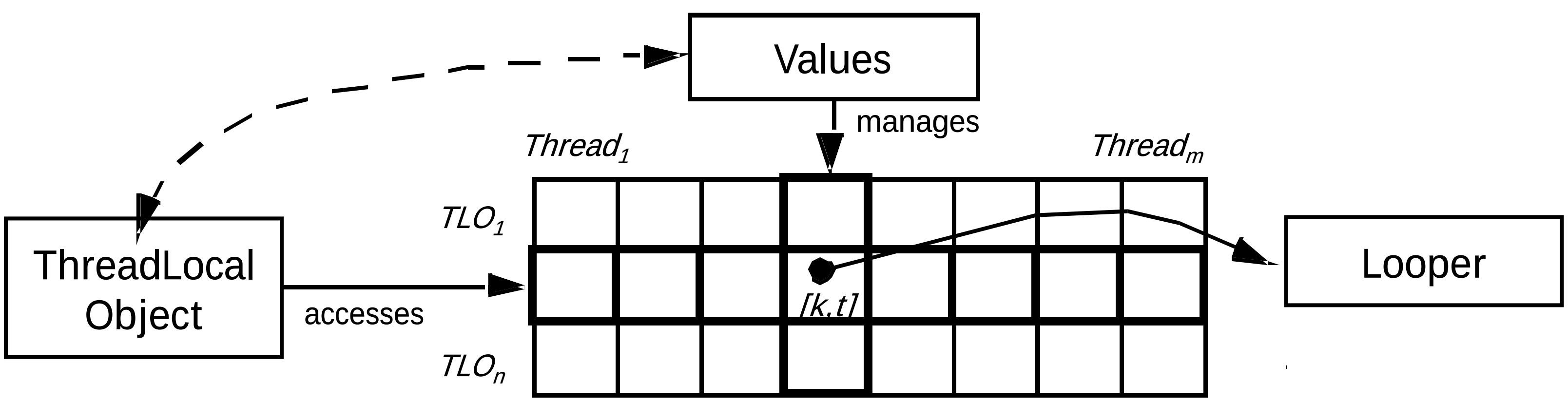
www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf has more info

Summary

- Android implements the *Thread-Specific Storage* pattern via the Java ThreadLocal class

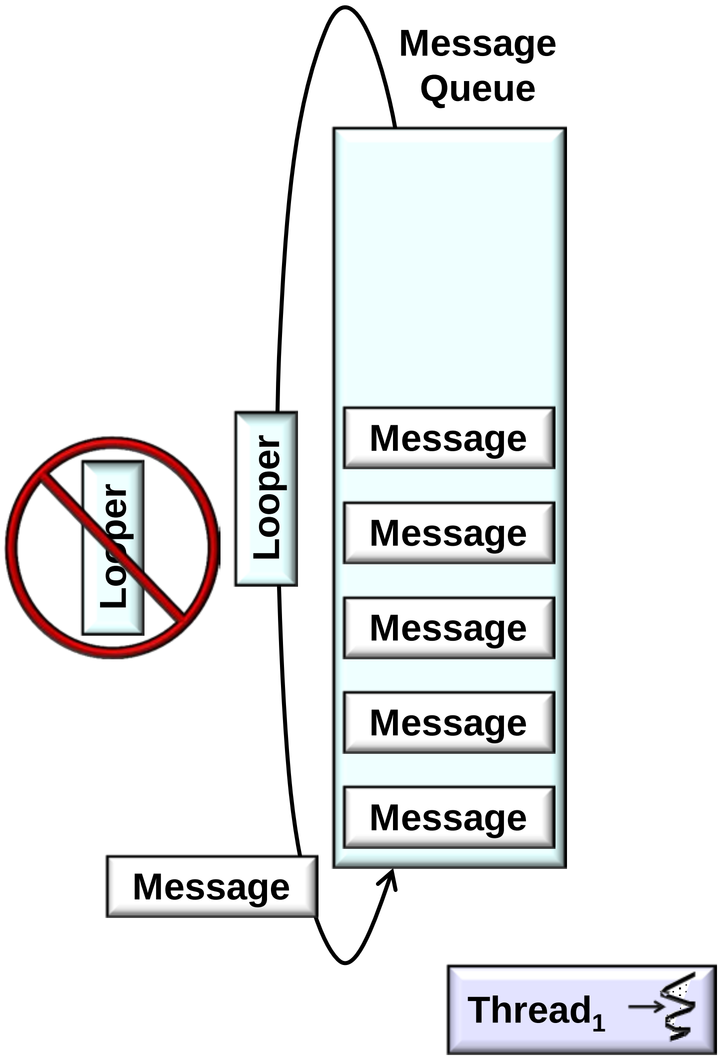
```
public class ThreadLocal<T> {
  ...
  public void set(T value) {
    ...
  }

  public T get() {
    ...
  }
```



Summary

- Android implements the *Thread-Specific Storage* pattern via the Java ThreadLocal class
- Android uses ThreadLocal to ensure a Thread has a single Looper



Summary

- Android implements the *Thread-Specific Storage* pattern via the Java ThreadLocal class
- Android uses ThreadLocal to ensure a Thread has a single Looper
- It’s also used in the constructor of Handler

```
public class Handler {
  ...
  public Handler() {
    mLooper = Looper.myLooper();
    if (mLooper == null)
      throw new RuntimeException
        ("Can't create handler
         inside thread that has
         not called Looper.
         prepare()");

    mQueue = mLooper.mQueue;
    mCallback = null;
  }
  ...
```