How to find deadlock not hitting it

Nikita Koval, Devexperts LLC

nkoval@devexperts.com
twitter.com/nkoval_

DEVEXPERTS

Real-life example

DEVEXPERTS

```
interface Con { // Connection
  void onRegister(ConnectionManager cm);
  void close();
}

class ConnectionManager {
  Set<Con> cons = new HashSet<>();

  synchronized void register(Con c) {
    cons.add(c);
    con.onRegister(this);
  }

  synchronized void unregister(Con c) {
    cons.remove(c);
  }

  ...
```


DEVEXPERTS

```
interface Con { // Connection
  void onRegister(ConnectionManager cm);
  void close();
}

class ConnectionManager {
  Set<Con> cons = new HashSet<>();

  synchronized void register(Con c) {
    cons.add(c);
    con.onRegister(this);
  }

  synchronized void unregister(Con c) {
    cons.remove(c);
  }

  ...
```

```
  // ... ConnectionManager

  synchronized int getConCount() {
    return cons.size();
  }

  synchronized void closeAll() {
    for (Con c: cons)
      c.close();
  }

}
```


DEVEXPERTS

```
class ConnectionImpl implements Con {
  ConnectionManager cm;

  @Override
  public synchronized void onRegister(ConnectionManager cm) {
    this.cm = cm;
  }

  @Override
  public synchronized void close() {
    if (cm != null) {
      cm.unregister(this);
      cm = null;
    }
  }
}
```

Run functional test…

Works well!

Run stress test…

DEVEXPERTS

```
public void test() throws Exception {
  ConnectionManager cm = new ConnectionManager();
  new Thread(() -> {
    while (true) {
      Connection con = new ConnectionImpl();
      cm.register(con);
      con.close(); // lock con -> lock cm
    }
  }).start();
  for (int i = 1; i < 100_000; i++) {
    System.out.println("Iteration #" + i);
    cm.closeAll(); // lock cm -> lock con
  }
}
```


DEVEXPERTS

```
Java stack information for the threads listed above:
===================================================
"Thread-0":
  at a.ConnectionManager.unregister(Sample.java:20)
  - waiting to lock <0x0000000795b8ee60> (a a.ConnectionManager)
  at a.ConnectionImpl.close(Sample.java:45)
  - locked <0x0000000795d72fd8> (a a.ConnectionImpl)
  at a.StressTest.lambda$test$0(StressTest.java:16)
  at a.StressTest$$Lambda$1/1297685781.run(Unknown Source)
  at java.lang.Thread.run(Thread.java:745)
"main":
  at a.ConnectionImpl.close(Sample.java:44)
  - waiting to lock <0x0000000795d72fd8> (a a.ConnectionImpl)
  at a.ConnectionManager.closeAll(Sample.java:29)
  - locked <0x0000000795b8ee60> (a a.ConnectionManager)
  at a.StressTest.test(StressTest.java:21) <27 internal calls>
Found 1 deadlock.
```


DEVEXPERTS

ConnectionManager cm
Connection con

```
1  MONITORENTER con
2  con.close()

3  MONITORENTER cm
4  cm.closeAll()

5  MONITORENTER cm
6  cm.unregister(con)

7  MONITORENTER con
8  con.close()
```


DEVEXPERTS

ConnectionManager cm
Connection con

1 MONITORENTER con	
2 con.close()	
	3 MONITORENTER cm
	4 cm.closeAll()
5 MONITORENTER cm	
6 cm.unregister(con)	
	7 MONITORENTER con
	8 con.close()

DEVEXPERTS

ConnectionManager cm
Connection con

```
1   MONITORENTER con
2   con.close()

3   MONITORENTER cm
4   cm.closeAll()

5   MONITORENTER cm
6   cm.unregister(con)

7   MONITORENTER con
8   con.close()
```


DEVEXPERTS

ConnectionManager cm
Connection con

```
1   MONITORENTER con
2   con.close()

                          3   MONITORENTER cm
                          4   cm.closeAll()
5   MONITORENTER cm
6   cm.unregister(con)

                          7   MONITORENTER con
                          8   con.close()
```


DEVEXPERTS

ConnectionManager cm
Connection con

```
1   MONITORENTER con
2   con.close()
```

```
3   MONITORENTER cm
4   cm.closeAll()
```

```
5   MONITORENTER cm
6   cm.unregister(con)
```

```
7   MONITORENTER con
8   con.close()
```


DEVEXPERTS

ConnectionManager cm
Connection con

1 MONITORENTER con	
2 con.close()	
	3 MONITORENTER cm
	4 cm.closeAll()
5 **MONITORENTER cm**	
6 cm.unregister(con)	
	7 **MONITORENTER con**
	8 con.close()

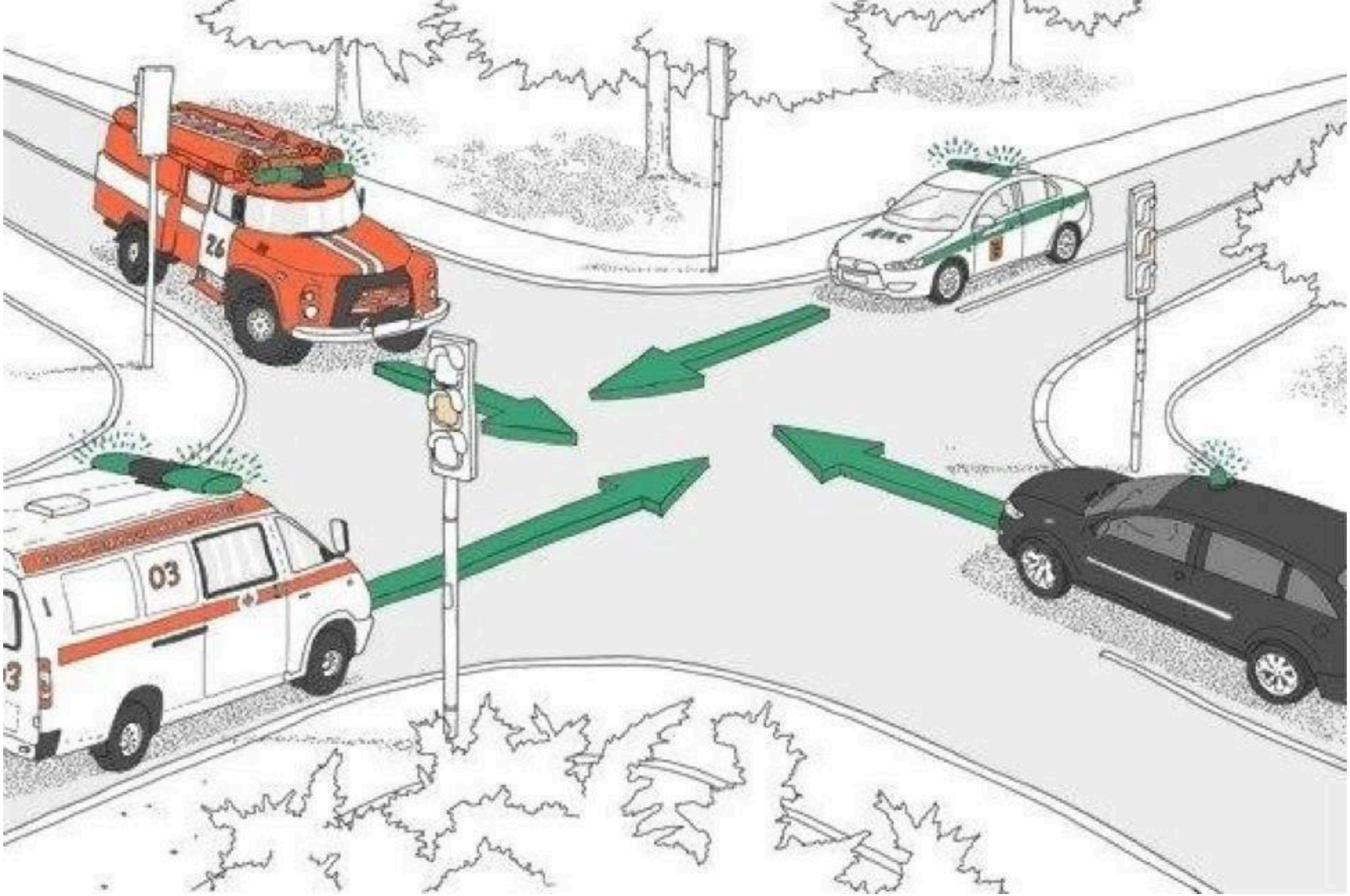
DEVEXPERTS

```
1   MONITO
2   con.clo

5   MONITORENTER cm
6   cm.unregister(con)
```

```
    ORENTER c
    cm.cl eAll()

7   MONITORENTER c
8   con.close()
```

HERE IS A DEADLOCK!

DEVEXPERTS


26
03

Program analysis approaches

- Static analysis
- Model checking
- Dynamic analysis

DEVEXPERTS

Static analysis

- Analyzes source code without its execution
- Can guarantee deadlock-freedom
- A lot of false positives

DEVEXPERTS

Model checking

- Analyzes program **model**
- Can guarantee deadlock-freedom
- No false positives
- Requires a lot of computational resources

DEVEXPERTS

Dynamic analysis

- Depends on the execution
- Few false positives
- Applicable for large programs

DEVEXPERTS

Dynamic analysis: techniques

- Collecting execution traces (off-line)
  - `JCarder`, `MulticoreSDK`
- **Detection immediately (on-line)**
  - All context information is available
    - Stacktrace, internal state
  - `VisualThreads`

DEVEXPERTS

ALGORITHM

Lock hierarchy

Any two locks should be acquired in the same order in the same thread

```
void transfer(Account from, Account to, int amount) {
  // Order lock acquisitions by ids
  if (from.id < to.id)
     lock(from, to)
   else
      lock(to, from)
   // Do transfer
  unlock(from, to)
}
```


DEVEXPERTS

Potential deadlock

- Lock hierarchy compliance – primary method to avoid deadlocks
- *Potential deadlock* – this hierarchy violation

DEVEXPERTS

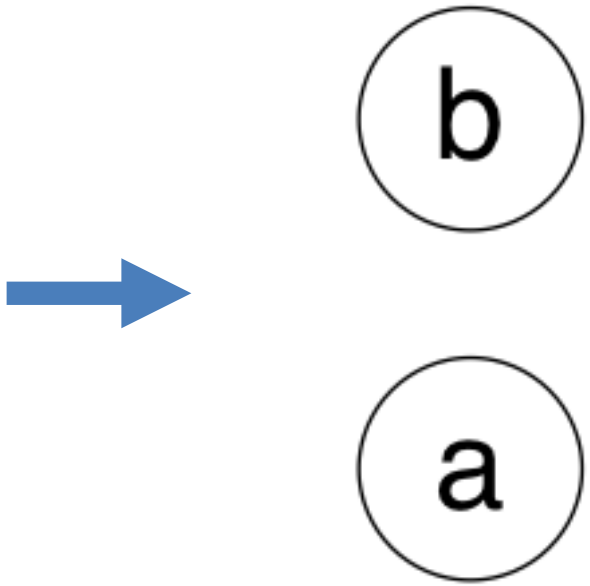
Lock graph

- Directed graph
- Vertex ↔ lock
- Edge (a, b) means that a is acquired before b
- Cycle = potential deadlock

DEVEXPERTS

Lock graph: example

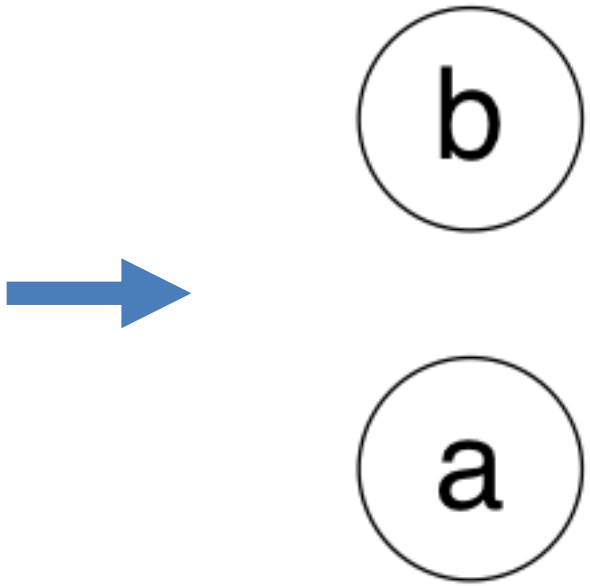
Thread 1	Thread 2
1 a.lock()	
2 b.lock()	
3 unlock(a, b)	
	4 b.lock()
	5 a.lock()
	6 unlock(a, b)



DEVEXPERTS

Lock graph: example

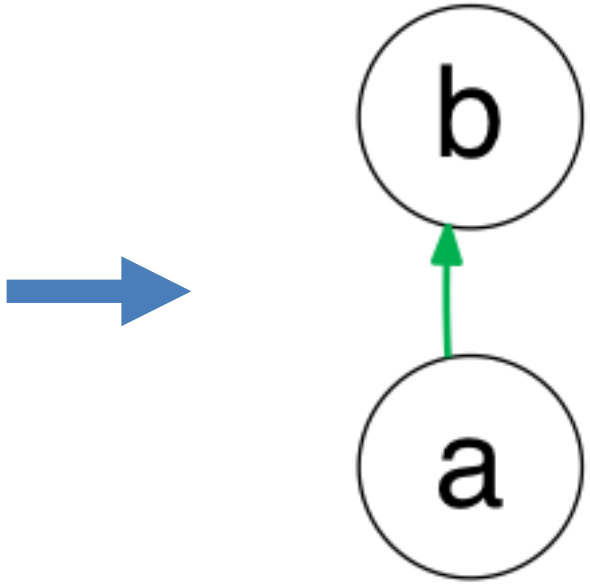
Thread 1	Thread 2
1 a.lock()	
2 b.lock()	
3 unlock(a, b)	
	4 b.lock()
	5 a.lock()
	6 unlock(a, b)



DEVEXPERTS

Lock graph: example

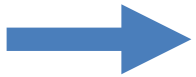
Thread 1	Thread 2
1 a.lock()	
2 **b.lock()**	
3 unlock(a, b)	
	4 b.lock()
	5 a.lock()
	6 unlock(a, b)

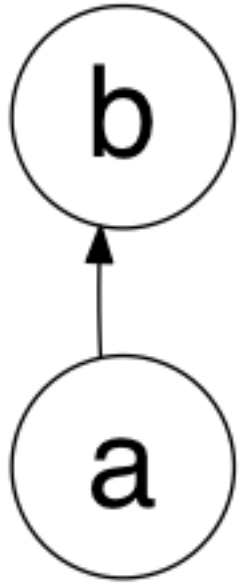


DEVEXPERTS

Lock graph: example

Thread 1	Thread 2
1 a.lock()	
2 b.lock()	
3 **unlock(a, b)**	
	4 b.lock()
	5 a.lock()
	6 unlock(a, b)

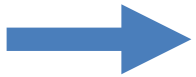


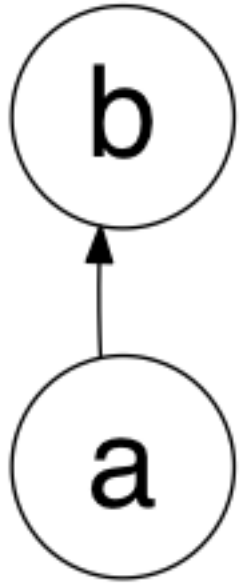


DEVEXPERTS

Lock graph: example

Thread 1	Thread 2
1 a.lock()	
2 b.lock()	
3 unlock(a, b)	
	4 b.lock()
	5 a.lock()
	6 unlock(a, b)

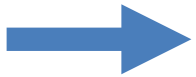


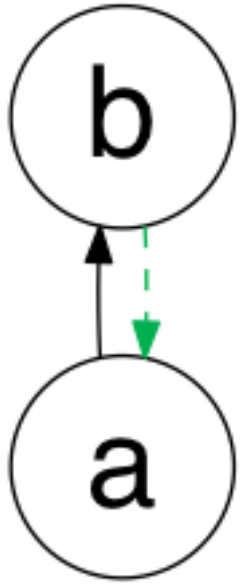


DEVEXPERTS

Lock graph: example

Thread 1	Thread 2
1 a.lock()	
2 b.lock()	
3 unlock(a, b)	
	4 b.lock()
	5 **a.lock()**
	6 unlock(a, b)



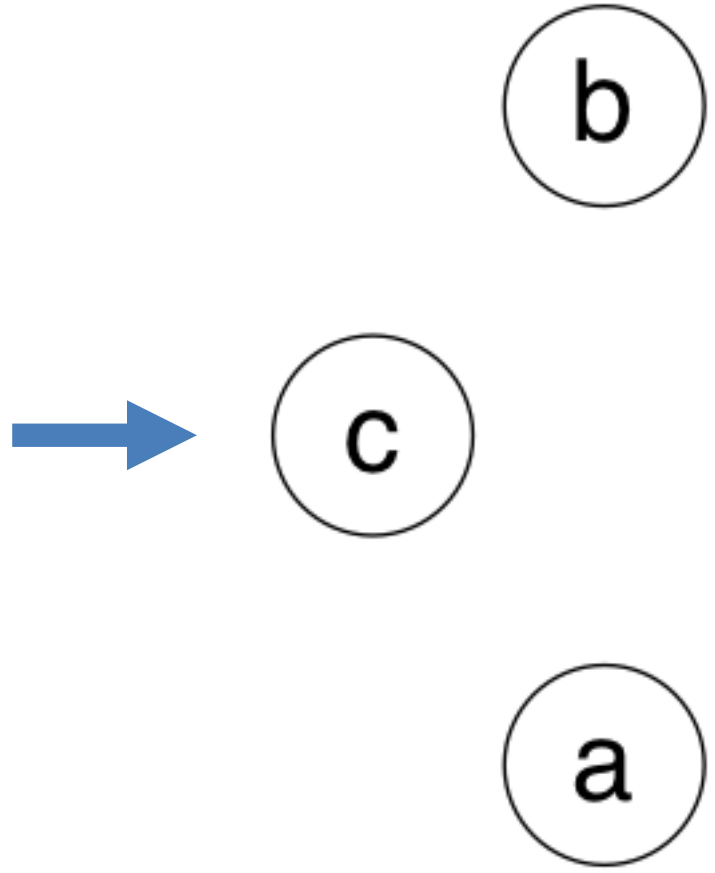


DEVEXPERTS

Minimization principle: problem

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 b.lock()	
4 unlock(a,b,c)	
	5 b.lock()
	6 a.lock()
	7 unlock(a,b)

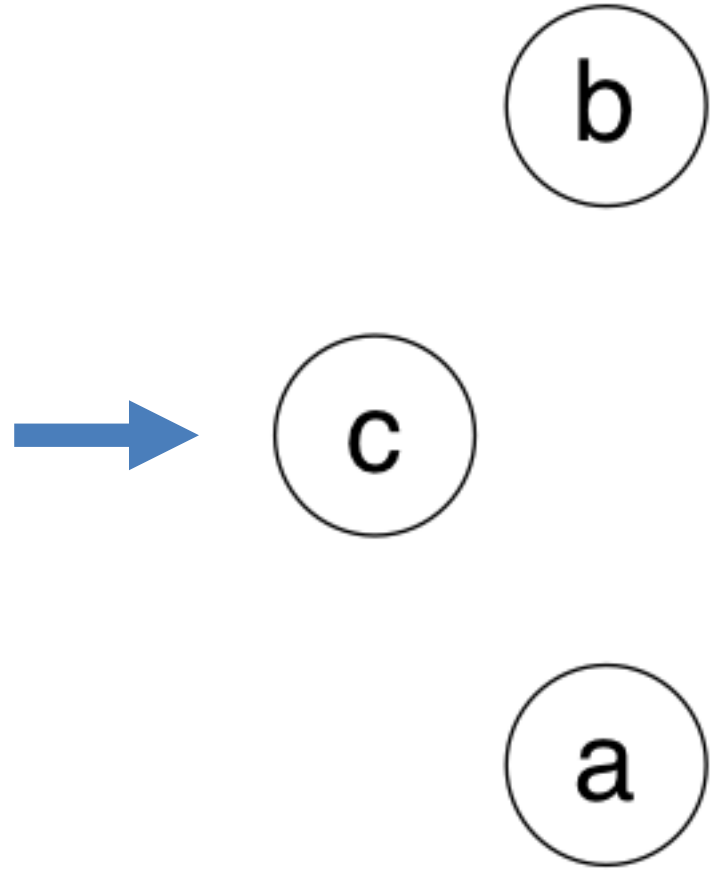


DEVEXPERTS

Minimization principle: problem (1)

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 b.lock()	
4 unlock(a,b,c)	
	5 b.lock()
	6 a.lock()
	7 unlock(a,b)

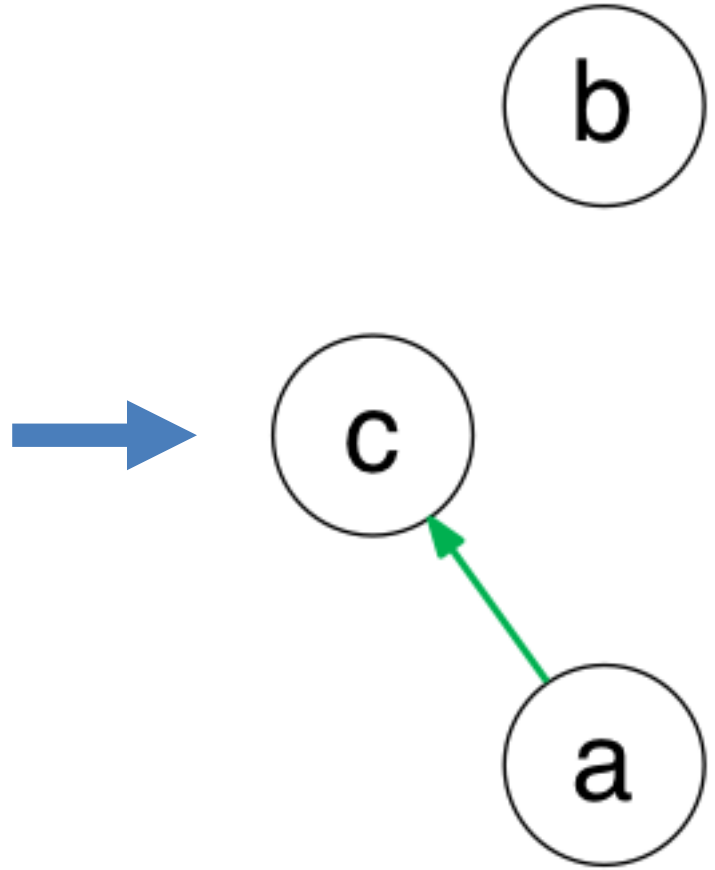


DEVEXPERTS

Minimization principle: problem (2)

- Two cycles may refer to the same error

	Thread 1	Thread 2
1	a.lock()	
2	**c.lock()**	
3	b.lock()	
4	unlock(a,b,c)	
5		b.lock()
6		a.lock()
7		unlock(a,b)

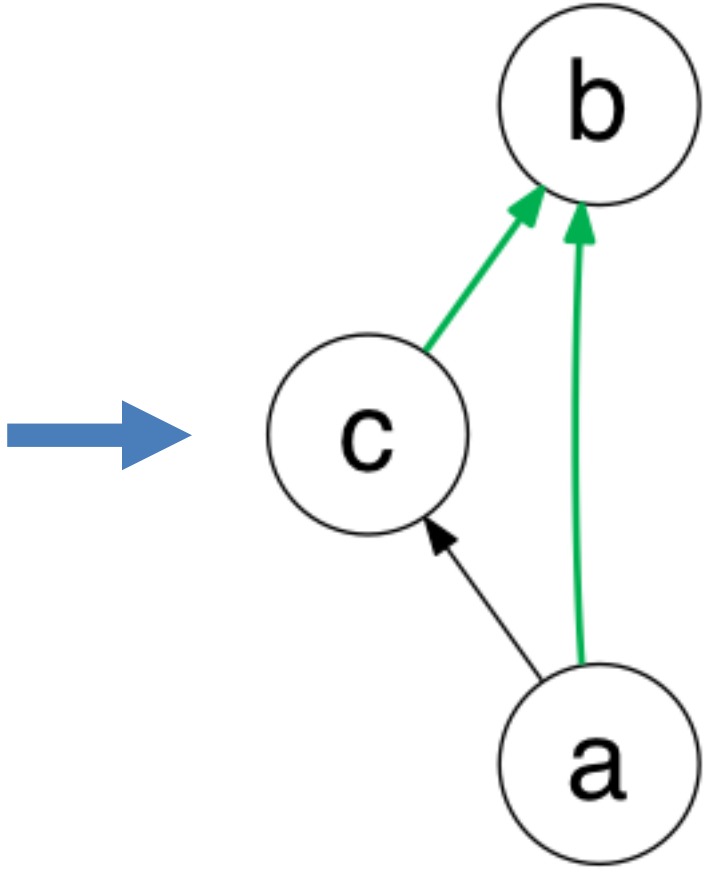


DEVEXPERTS

Minimization principle: problem (3)

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 **b.lock()**	
4 unlock(a,b,c)	
	5 b.lock()
	6 a.lock()
	7 unlock(a,b)

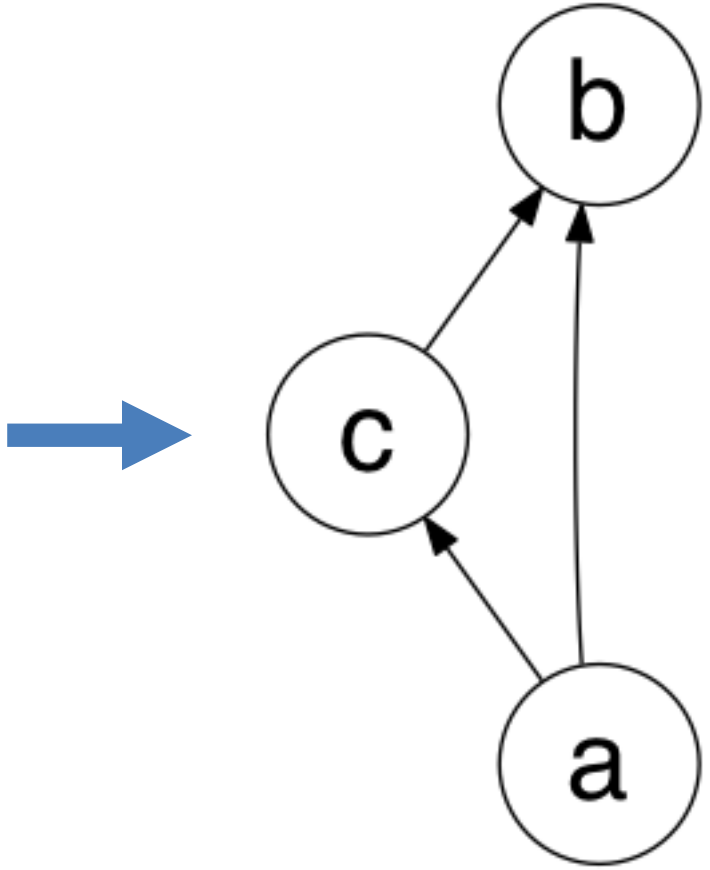


DEVEXPERTS

Minimization principle: problem (4)

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 b.lock()	
4 **unlock(a,b,c)**	
	5 b.lock()
	6 a.lock()
	7 unlock(a,b)

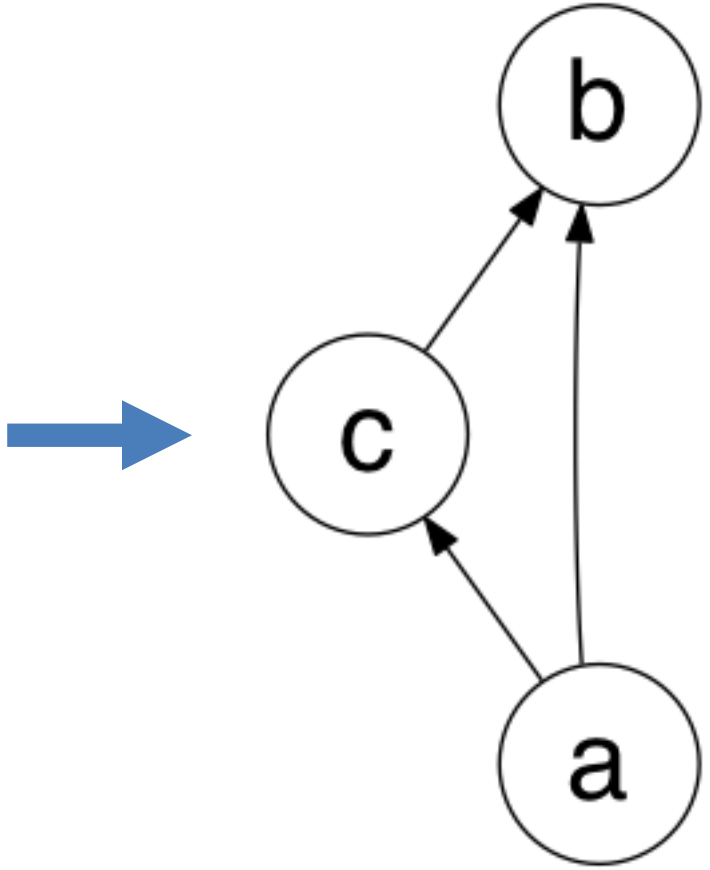


DEVEXPERTS

Minimization principle: problem (5)

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 b.lock()	
4 unlock(a,b,c)	
	5 b.lock()
	6 a.lock()
	7 unlock(a,b)

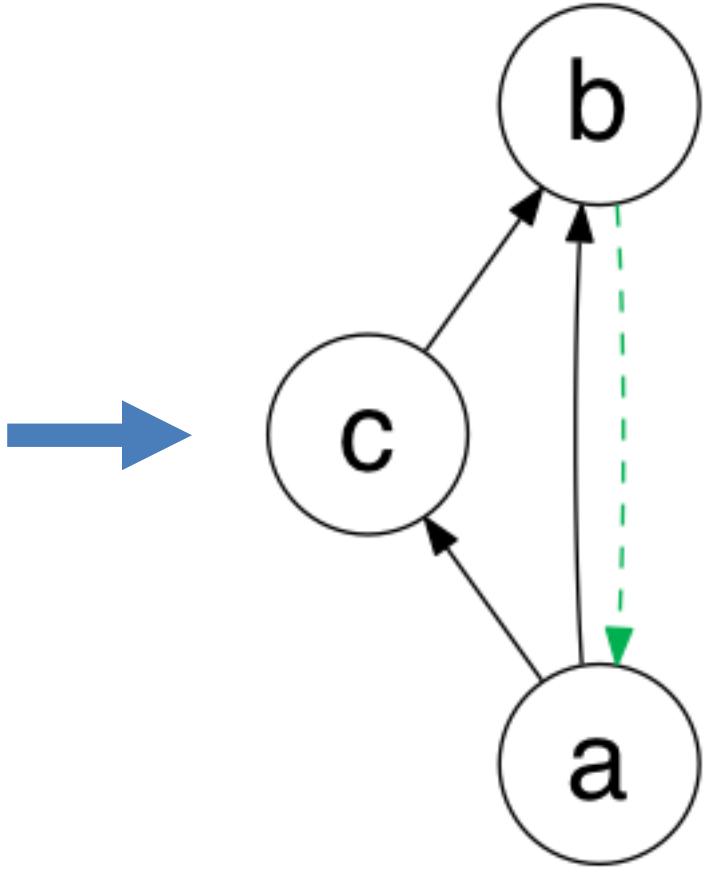


DEVEXPERTS

Minimization principle: problem (6)

- Two cycles may refer to the same error

Thread 1	Thread 2
1 a.lock()	
2 c.lock()	
3 b.lock()	
4 unlock(a,b,c)	
	5 b.lock()
	6 **a.lock()**
	7 unlock(a,b)



DEVEXPERTS

Minimization principle

- Shorter cycle is more useful
- Notify about the shortest one only

DEVEXPERTS

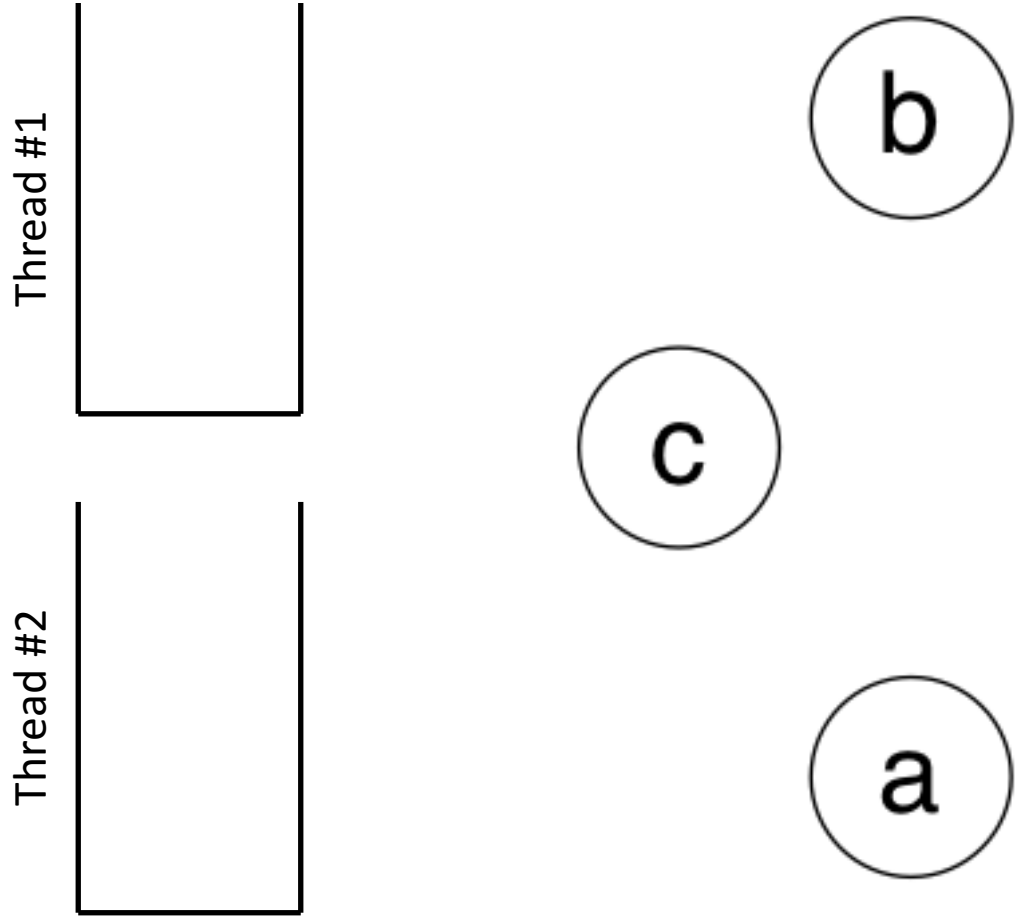
Algorithm layout

- Capture lock acquire and release operations
- On lock acquisition:
  - Add lock to the set of locks held by the current thread
  - Add new edges to the lock graph
  - Report new cycles
- On lock release:
  - Remove lock from the set of held locks

DEVEXPERTS

Lock acquisition and release analysis

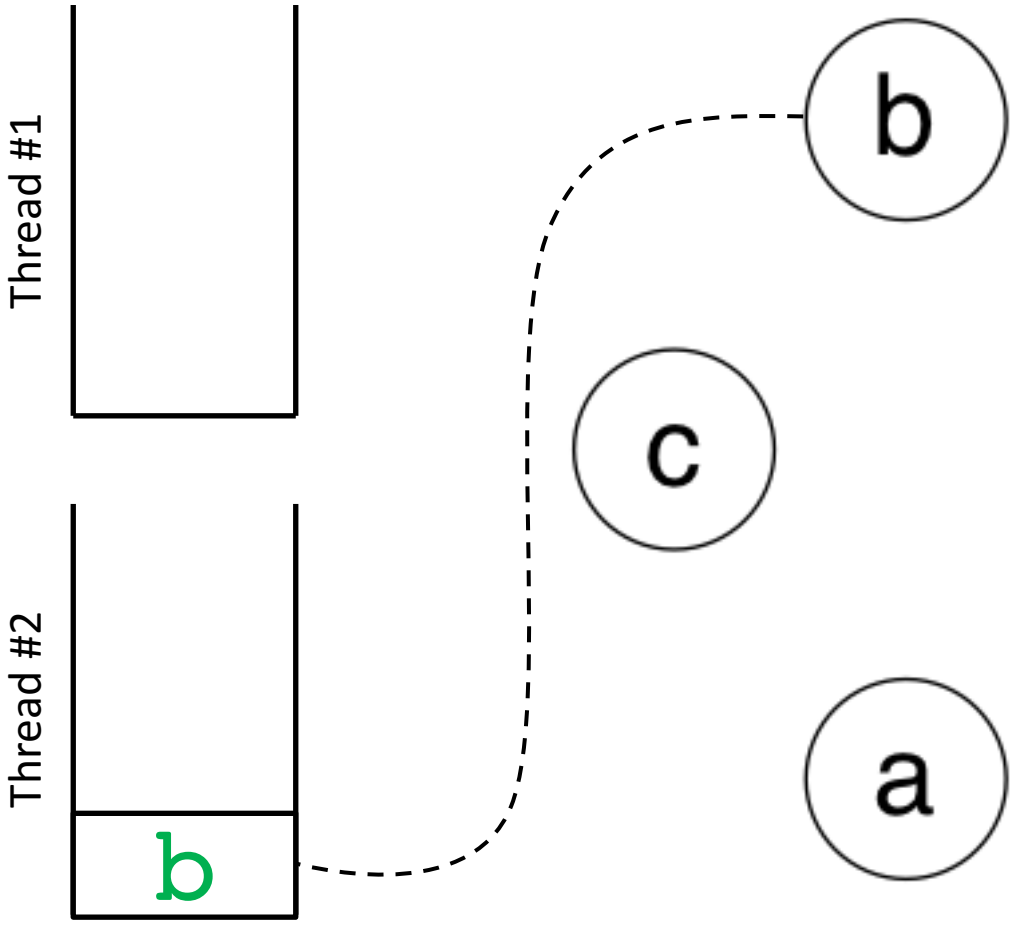
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (1)

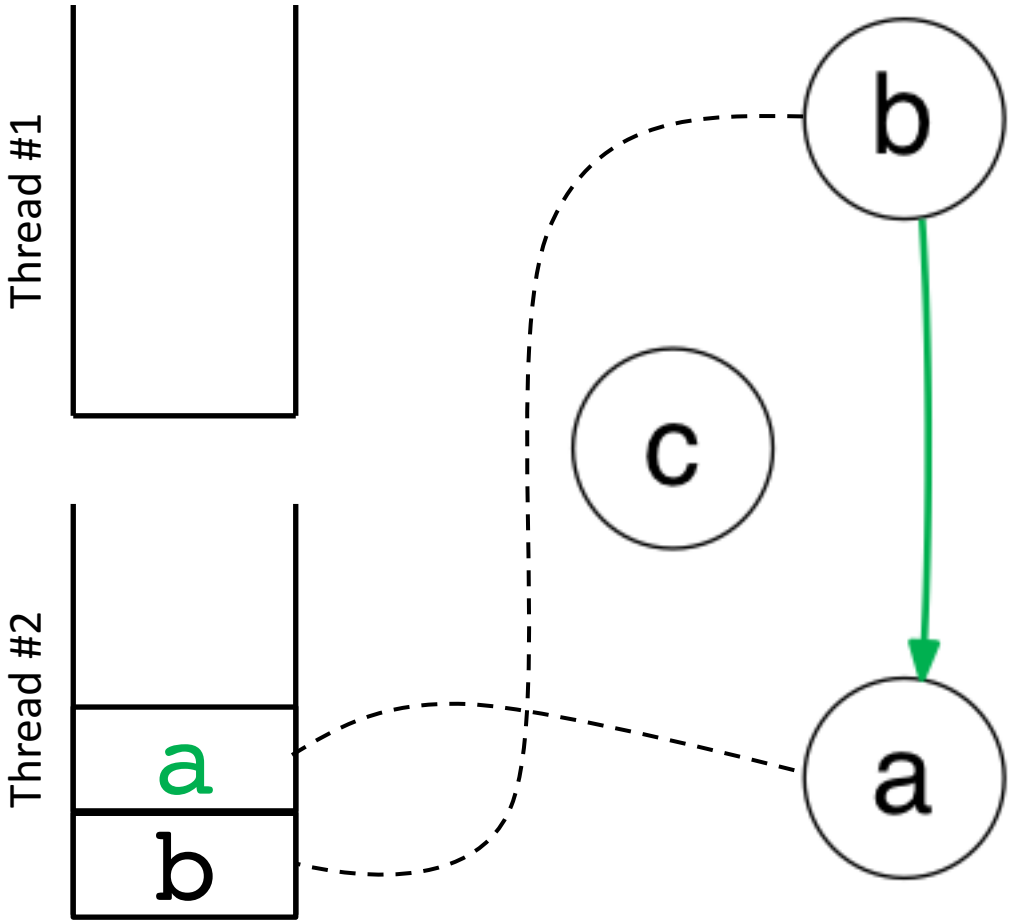
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (2)

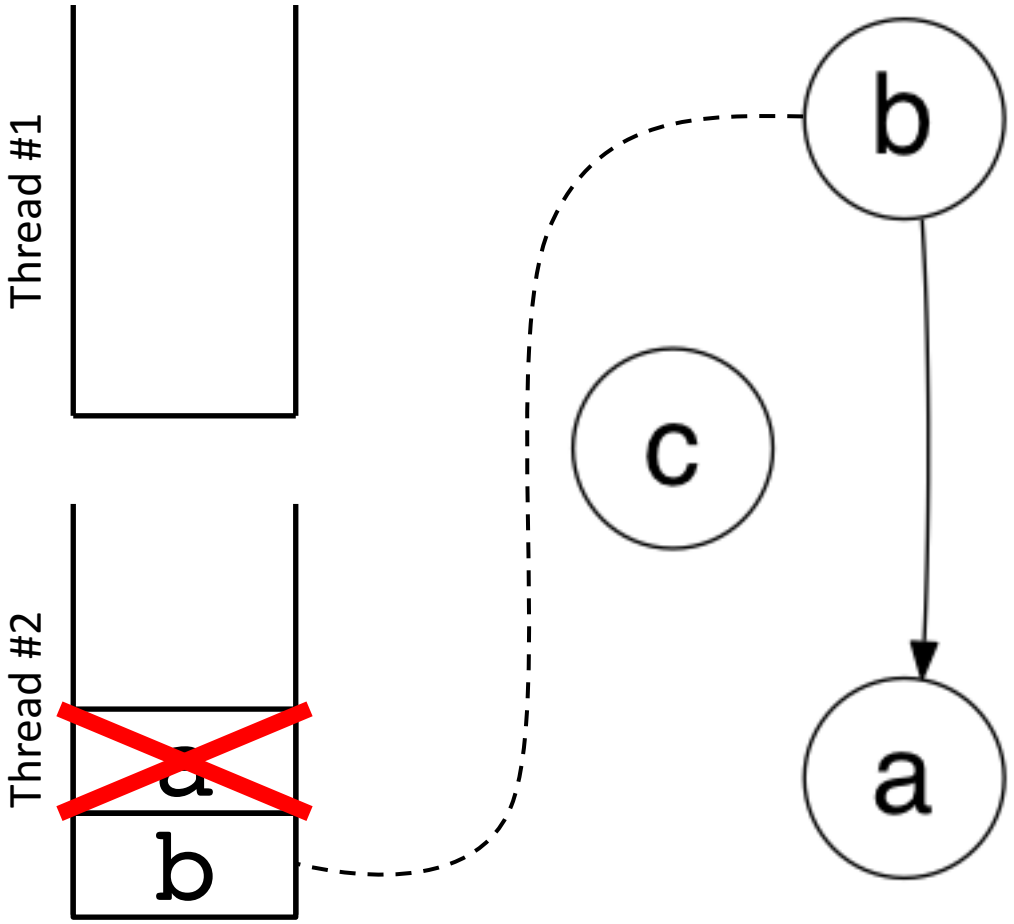
Thread 1	Thread 2
	1 b.lock()
	2 **a.lock()**
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (3)

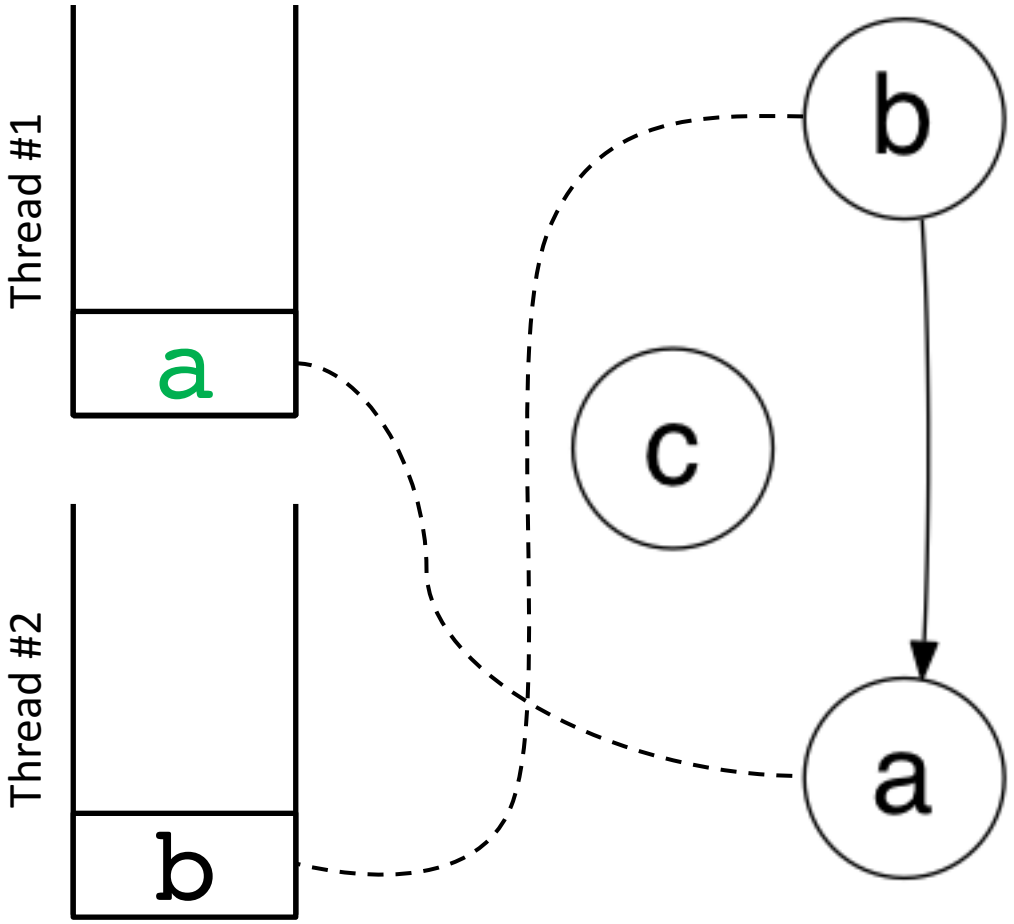
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 **a.unlock()**
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (4)

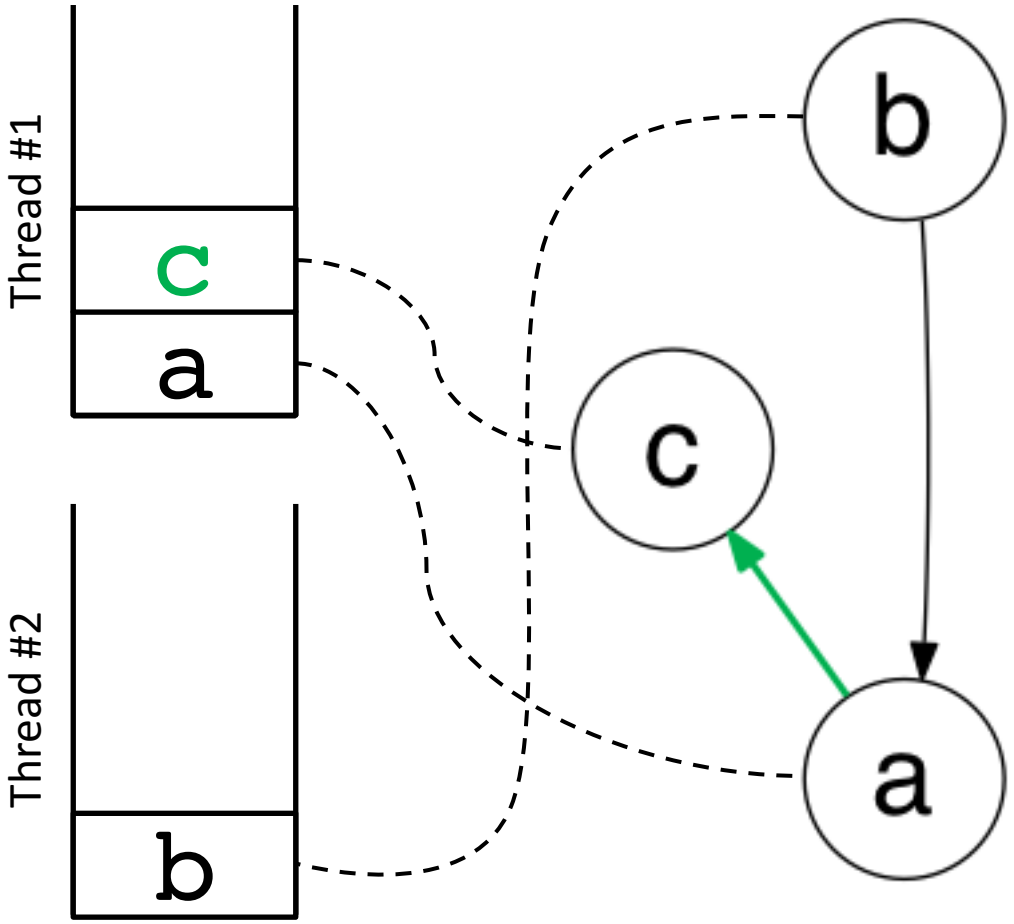
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 **a.lock()**	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (5)

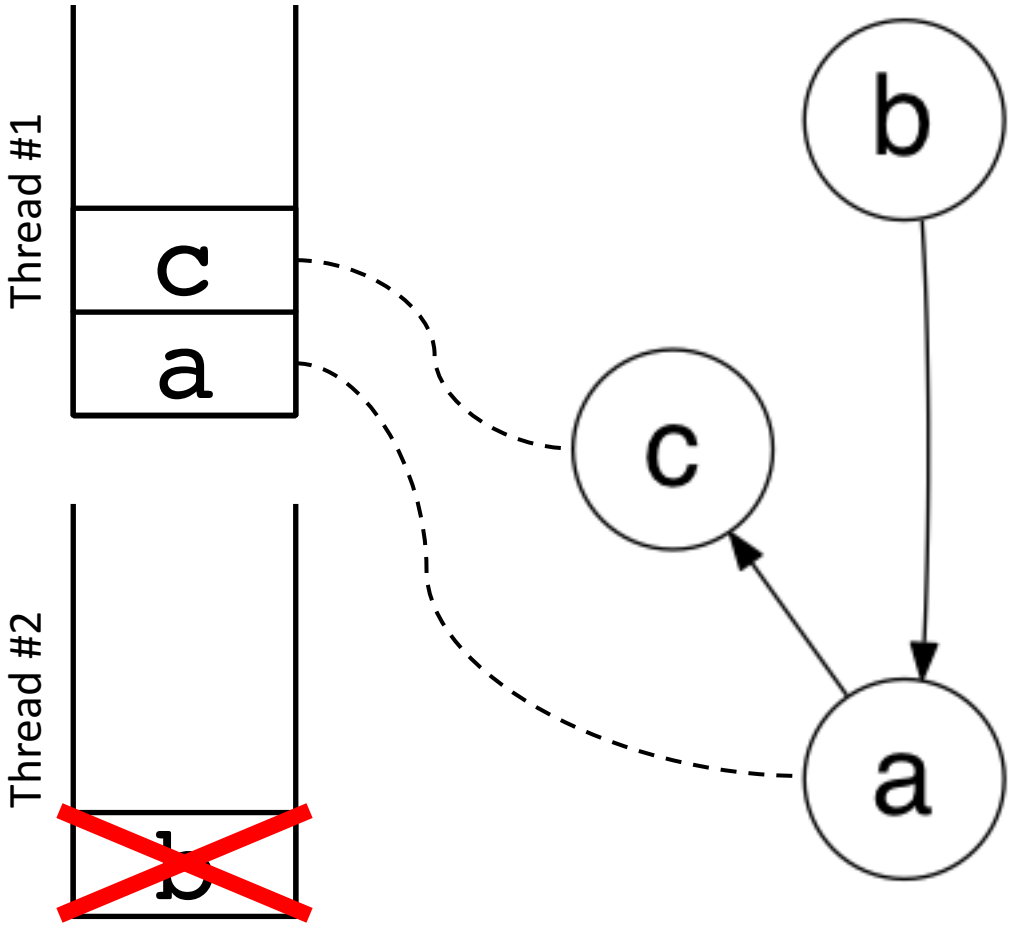
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 **c.lock()**	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (6)

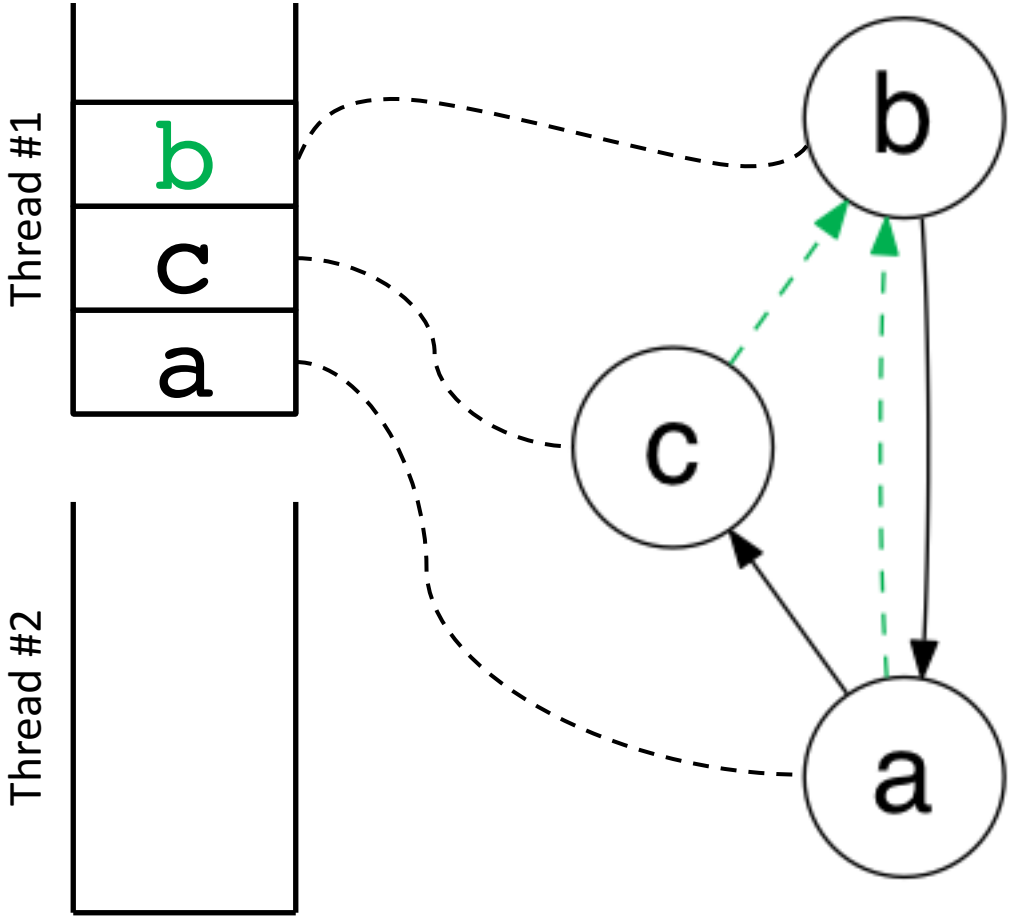
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (7)

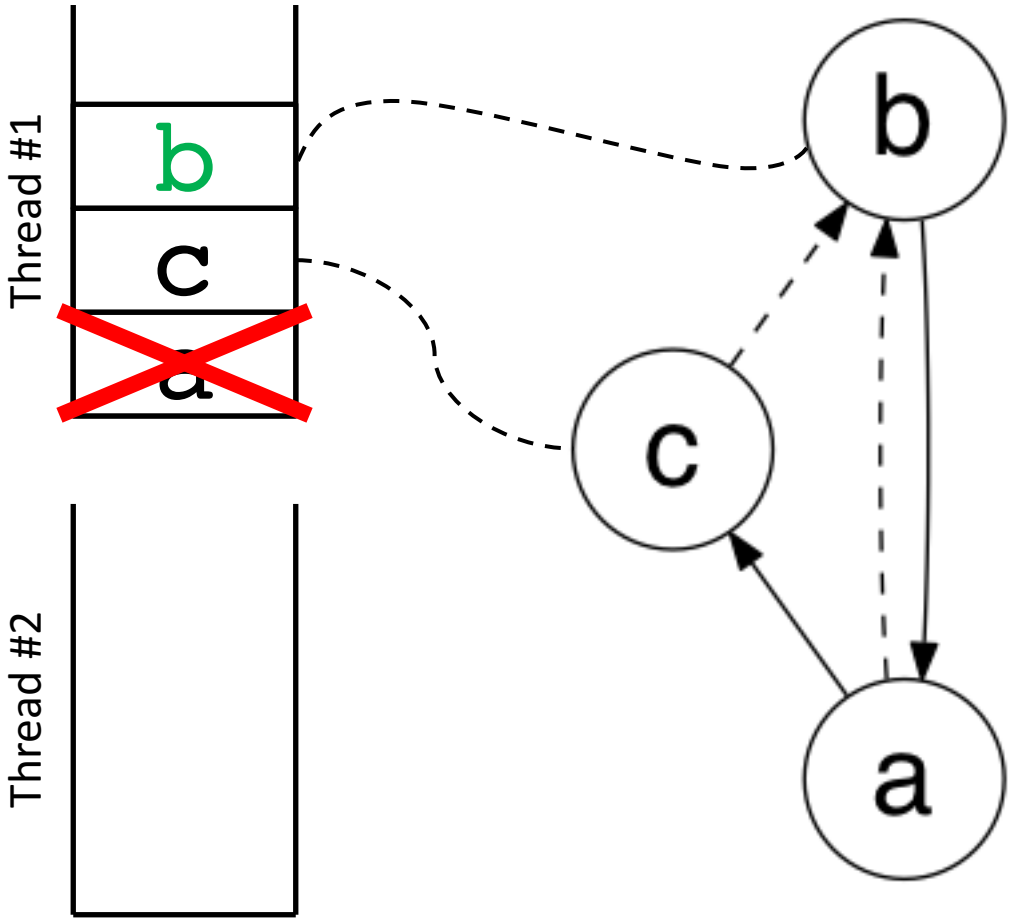
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 **b.lock()**	
8 a.unlock()	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (8)

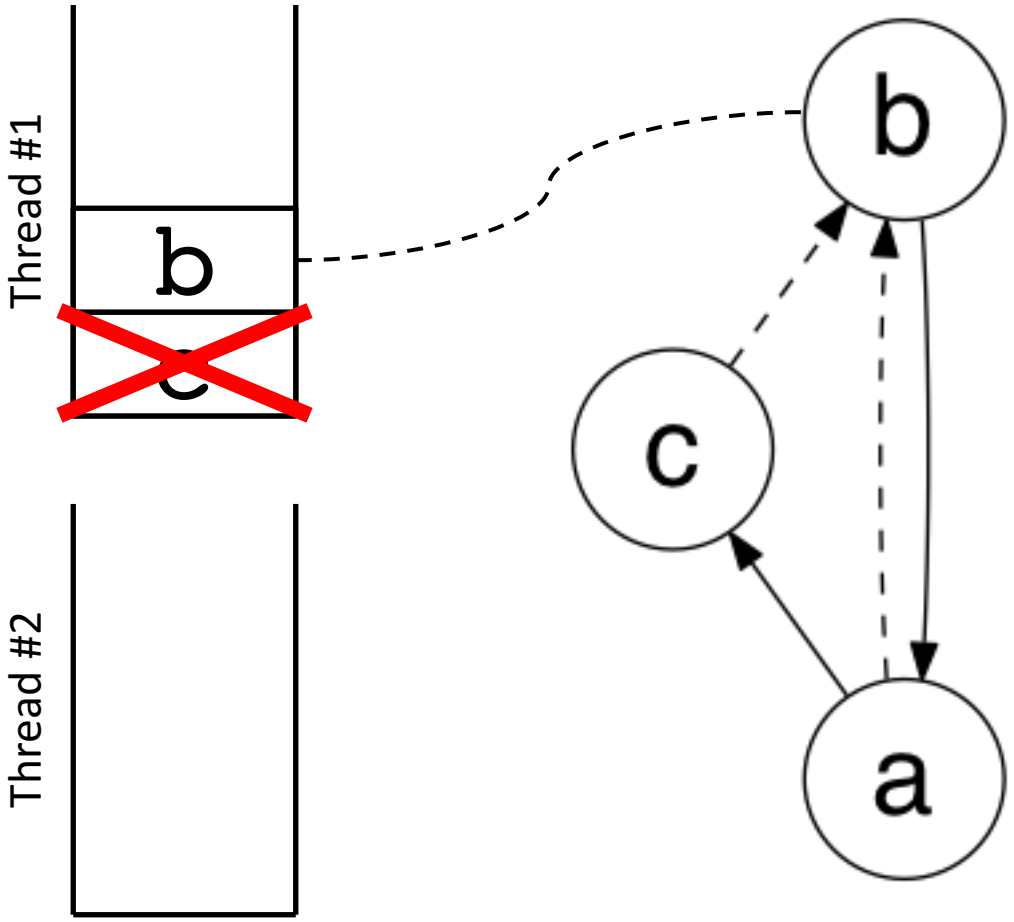
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 **a.unlock()**	
9 c.unlock()	
10 b.unlock()	



DEVEXPERTS

Lock acquisition and release analysis (9)

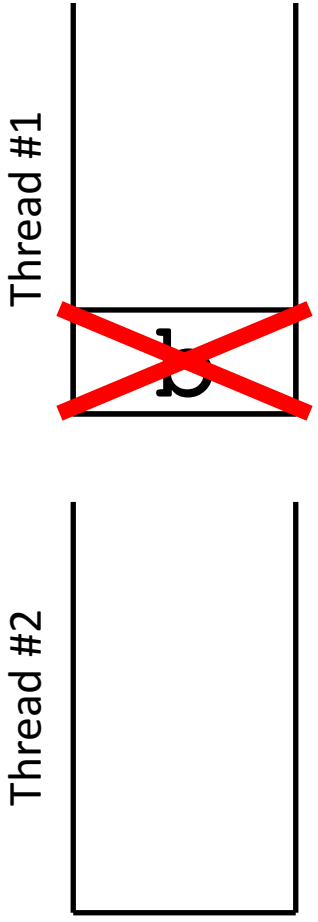
Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 **c.unlock()**	
10 b.unlock()	

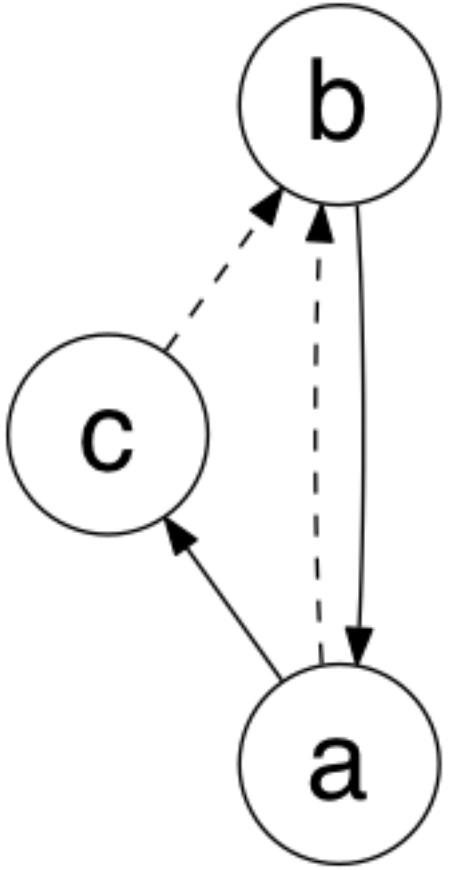


DEVEXPERTS

Lock acquisition and release analysis (10)

Thread 1	Thread 2
	1 b.lock()
	2 a.lock()
	3 a.unlock()
4 a.lock()	
5 c.lock()	
	6 b.unlock()
7 b.lock()	
8 a.unlock()	
9 c.unlock()	
10 **b.unlock()**	





DEVEXPERTS

Set of held locks

- Common lock usage patterns

```
synchronized(o) {
  // Do something
}

synchronized void f() {
  // Do something
}
```

```
l.lock()
try {
  // Do something
} finally {
  l.unlock()
}
```

- Locks are acquired and released in LIFO (stack) order

DEVEXPERTS

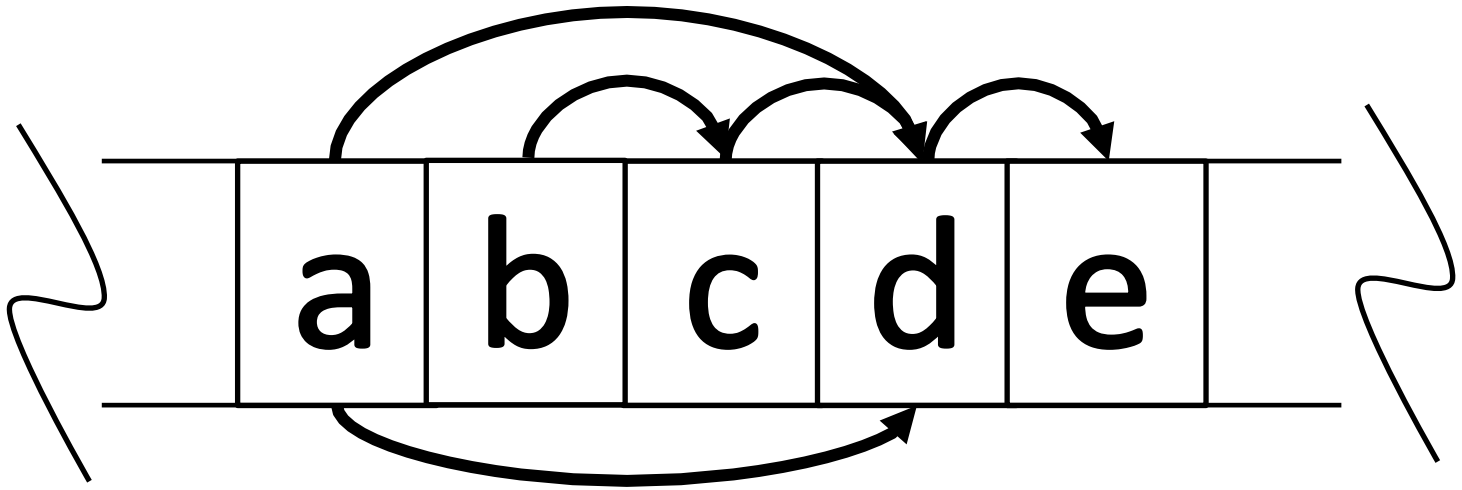
Set of held locks

- Stack with possibility of removing from the middle
- Commonly works in $O(1)$
- Works in $O(B)$ at worst
  - B – number of locks acquired by the current thread

DEVEXPERTS

Topological order

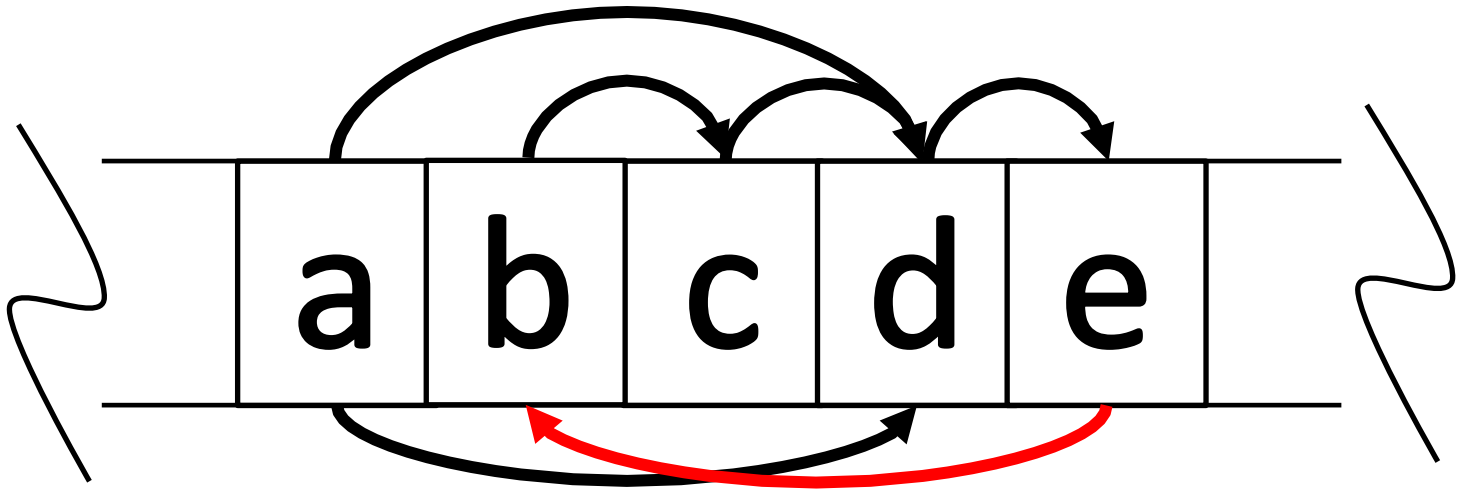
- Constructed by the edges of the directed graph
- No topological order ⇔ graph has a cycle



DEVEXPERTS

Topological order

- Constructed by the edges of the directed graph
- No topological order ⇔ graph has a cycle



DEVEXPERTS

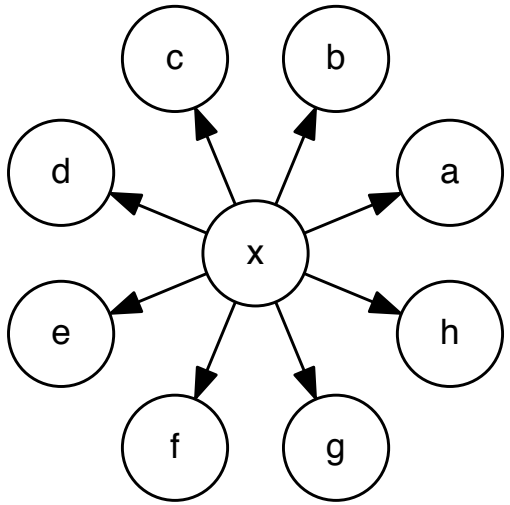
Topological order

- The acyclic part of the lock graph is to be kept
- Edge $(\boldsymbol{a}, \boldsymbol{b})$ leads to a topological order violation $\Rightarrow$ the shortest path $\boldsymbol{b} \rightsquigarrow \boldsymbol{a}$ refers to the shortest cycle
- Incremental maintenance
  - MNR algorithm suggested by Marchetti-Spaccamela et al.

DEVEXPERTS

New nodes processing

- Typical pattern in lock graphs

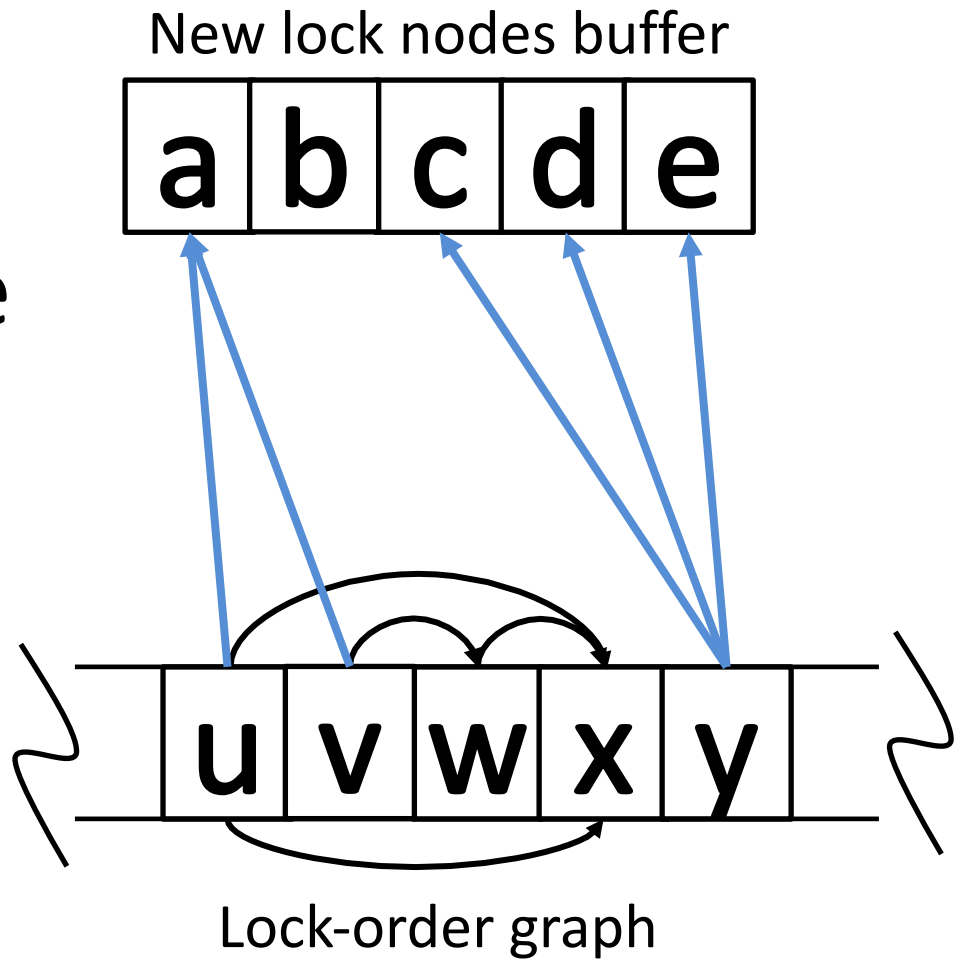


- One long-lived lock at the center and many hundreds of short-lived locks around it

DEVEXPERTS

New nodes processing

- Lock-free buffer for new nodes
- Initialize topological order value
  - if buffer is full
  - when outgoing edge should be added



DEVEXPERTS

Complexity

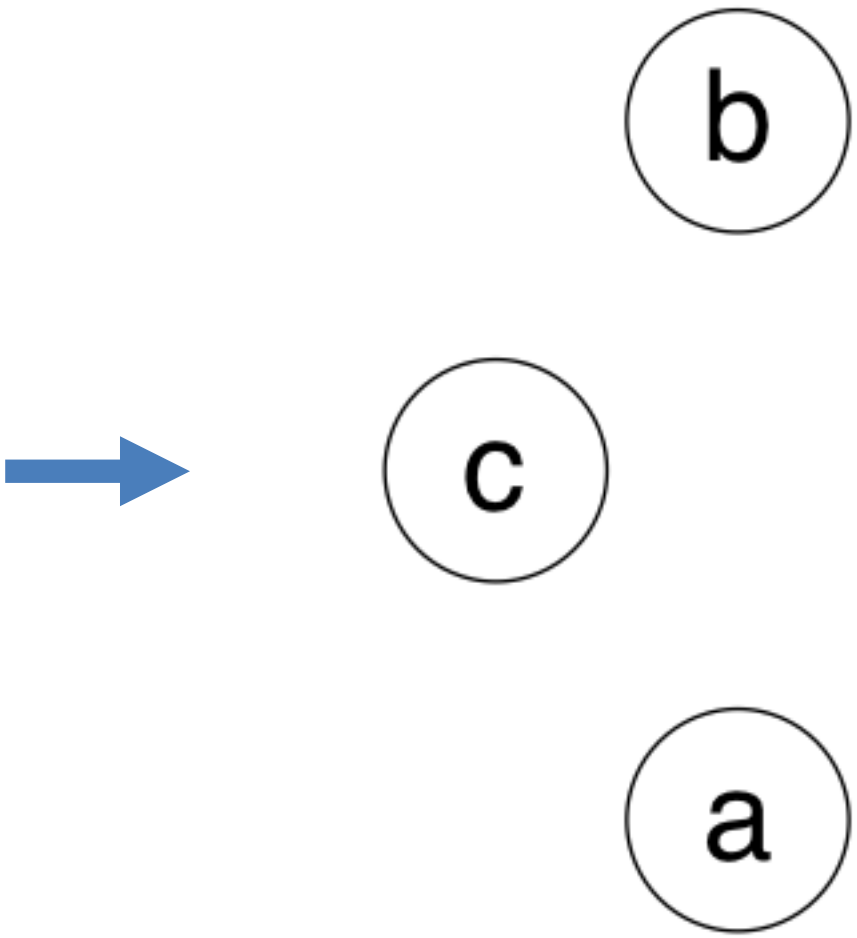
- Lock acquisition analysis
  - If acquisition does not produce new cycles
    - Typical case
    - $O(B + |V| + |E|)$ at worst
  - New cycle is detected
    - $O\big(B \cdot (|V| + |E|)\big)$
- Lock release analysis
  - $O(B)$ at worst

DEVEXPERTS

Limitation: cycle minimization

```
void transfer(Account from,
              Account to,
              int amount) {
  from.lock()
  to.lock()
  // Do transfer
  unlock(from, to)
}
```

```
1  transfer(a, c, …)
2  transfer(c, b, …)
3  transfer(b, a, …)
4  transfer(a, b, …)
```

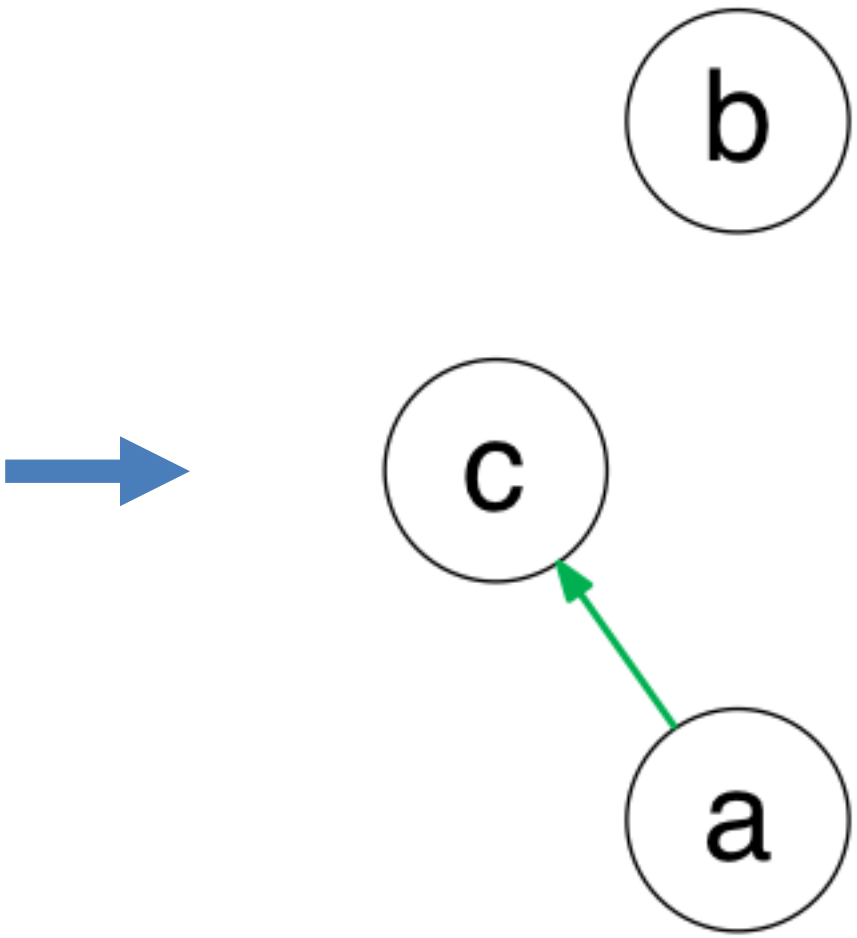


DEVEXPERTS

Limitation: cycle minimization (1)

```
void transfer(Account from,
              Account to,
              int amount) {
  from.lock()
  to.lock()
  // Do transfer
  unlock(from, to)
}
```

```
1  transfer(a, c, …)
2  transfer(c, b, …)
3  transfer(b, a, …)
4  transfer(a, b, …)
```

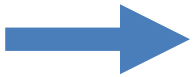


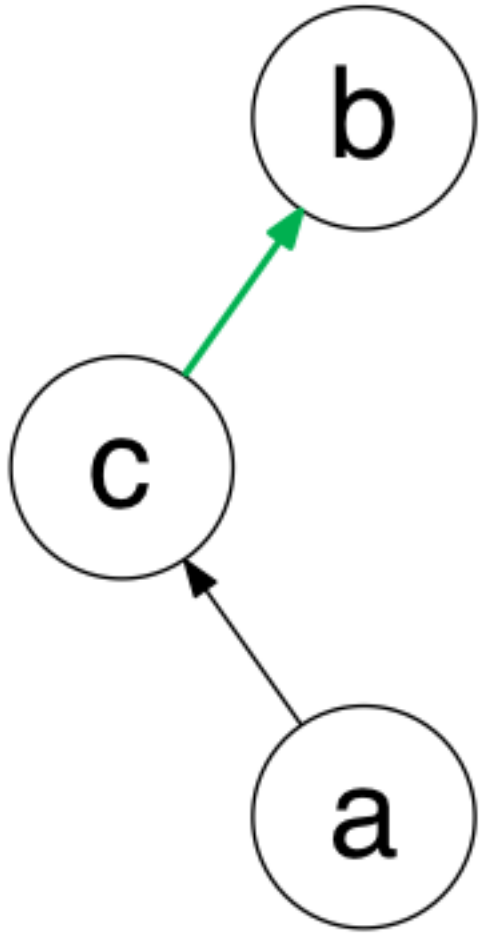
DEVEXPERTS

Limitation: cycle minimization (2)

```
void transfer(Account from,
              Account to,
              int amount) {
  from.lock()
  to.lock()
  // Do transfer
  unlock(from, to)
}
```

```
1  transfer(a, c, …)
2  transfer(c, b, …)
3  transfer(b, a, …)
4  transfer(a, b, …)
```



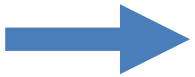


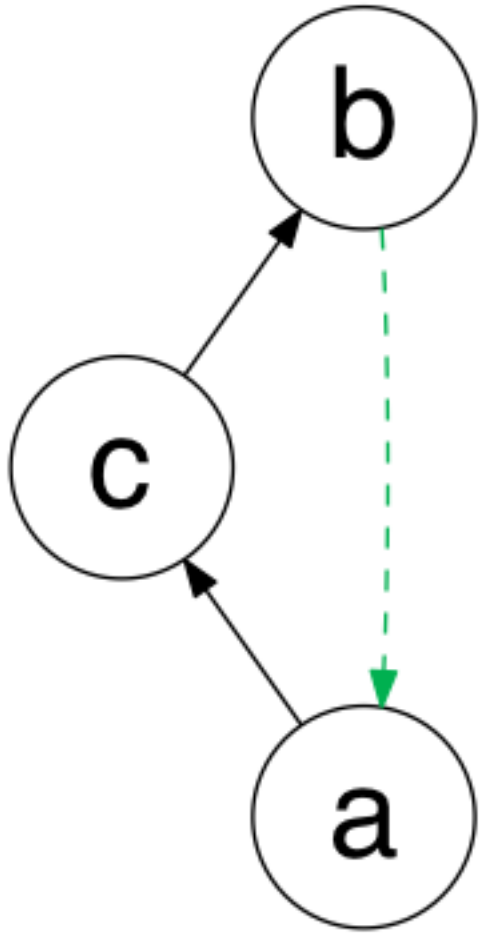
DEVEXPERTS

Limitation: cycle minimization (3)

```
void transfer(Account from,
              Account to,
              int amount) {
  from.lock()
  to.lock()
  // Do transfer
  unlock(from, to)
}
```

```
1  transfer(a, c, …)
2  transfer(c, b, …)
3  transfer(b, a, …)
4  transfer(a, b, …)
```



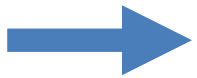


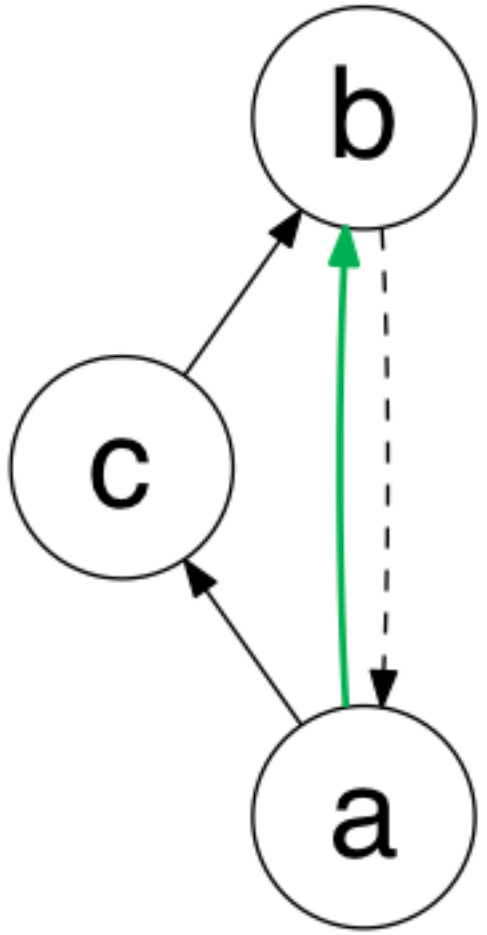
DEVEXPERTS

Limitation: cycle minimization (4)

```
void transfer(Account from,
              Account to,
              int amount) {
  from.lock()
  to.lock()
  // Do transfer
  unlock(from, to)
}
```

```
1  transfer(a, c, …)
2  transfer(c, b, …)
3  transfer(b, a, …)
4  transfer(a, b, …)
```





DEVEXPERTS

Limitation: cycle minimization

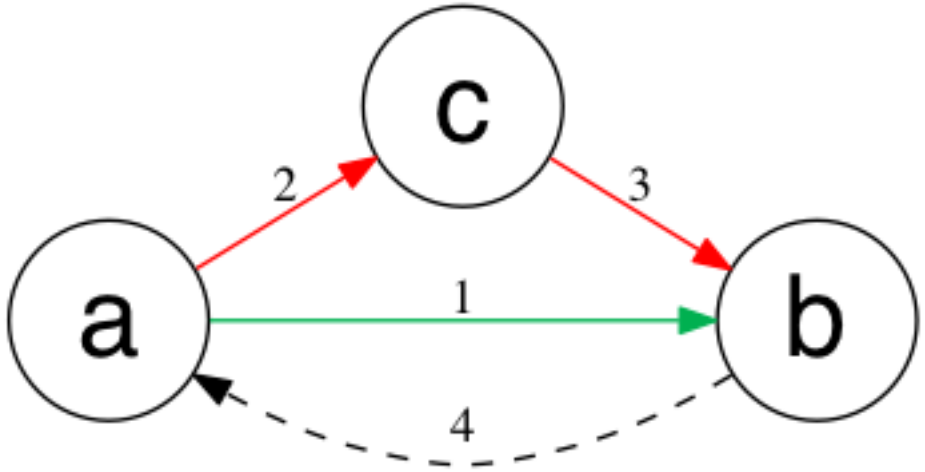
- Adding edge (a, b) creates new cycle
- Minimization of $\langle b, \ldots, a \rangle \Leftrightarrow$ minimization of $b \rightsquigarrow a$
  - Can be implemented with help of BFS

- Usually 2 or 3 vertexes in cycle
- Current implementation doesn't support minimization

DEVEXPERTS

Limitation: independent cycles

- Logically independent cycles:
  - $\langle a, c, b \rangle$
  - $\langle a, b \rangle$

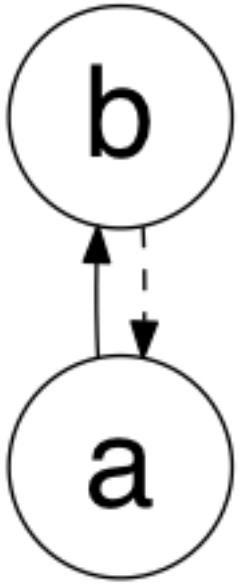


- Independent cycle is ignored due to minimization principle
- One useful cycle is better ☺

DEVEXPERTS

Limitation: single threaded cycle

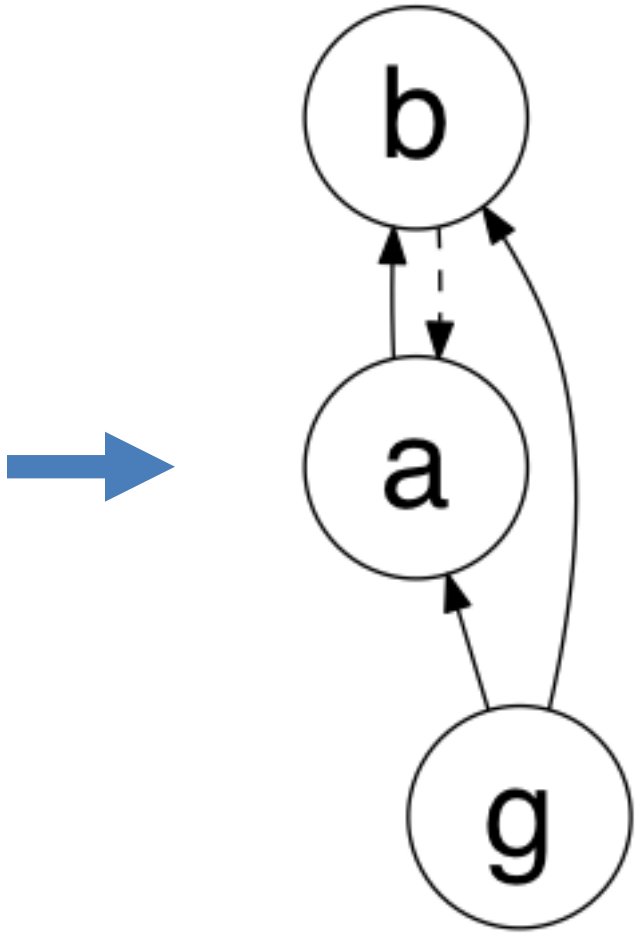
```
    Lock a, b
-------------------
1   a.lock()
2   b.lock()
3   unlock(a, b)

4   b.lock()
5   a.lock()
6   unlock(a, b)
```



DEVEXPERTS

Limitation: guarded cycle

Thread 1	Thread 2
1 **g.lock()**	
2 a.lock()	
3 b.lock()	
4 unlock(a, b)	
5 **g.unlock()**	
	6 **g.lock()**
	7 b.lock()
	8 a.lock()
	9 unlock(a, b)
	10 **g.unlock()**



DEVEXPERTS

IMPLEMENTATION

Dl-Check

- https://github.com/Devexperts/dlcheck
- Handy for testing
  - Throws an exception if `-Ddlcheck.fail=true`
  - Or write your own behavior
- Implemented as a Java agent
  - `java -javaagent:dlcheck.jar -jar your_app.jar`
  - Modifies byte-code within `ASM` framework

DEVEXPERTS

Byte-code transformation

Lock acquisition and release analysis:

- First and last action in `synchronized` method
- `MONITORENTER` and `MONITOREXIT` instructions
- `java.util.concurrent.locks.Lock`
  - Method `lock`
  - Successful invocation of method `tryLock`
  - Method `unlock`

DEVEXPERTS

Synchronized method: in simple terms

- Lock acquisition analysis
- Lock release analysis

```
synchronized void f() {
  // Do something
}
```

```
synchronized void f() {
  onLockAcquire(this,
                2 /*location id*/);
  try {
    // Do something
  } finally {
    onLockRelease(this);
  }
}
```


DEVEXPERTS

Synchronized meth

What about… exceptions?

- Lock acquisition analysis
- Lock release an

```
synchronized
   // Do some
}
```

```
                                    () {
   onLockAcquire(this,
                          2 /*location id*/);
   try {
      // Do something
   } finally {
      onLockRelease(this);
   }
}
```


DEVEXPERTS

Synchronized method: exception

WRONG

```
synchronized void f() {
  onLockAcquire(this, 2);
  try {
    // Do something
  } finally {
    onLockRelease(this);
  }
}
```

CORRECT

```
synchronized void f() {
  try {
    onLockAcquire(this, 2);
    // Do something
  } finally {
    onLockRelease(this);
  }
}
```


DEVEXPERTS

Synchronized method: byte-code

- Lock acquisition analysis
- Lock release analysis

```
synchronized void g() {
  try {
    onLockAcquire(this, 2);
    // Do something
  } finally {
    onLockRelease(this);
  }
}
```

```
synchronized void g();
Code:
   0: aload_0 // this
   1: iconst_2 // location id
   2: invokestatic #22 // onLockAcquire
      // Do something
   5: aload_0 // this
   6: invokestatic #26 // onLockRelease
   9: return
  10: aload_0 // this
  11: invokestatic #26 // onLockRelease
  14: athrow
Exception table:
  from  to  target  type
     0  9       10   any
```


DEVEXPERTS

Synchronized block

```
void f() {
  synchronized (o) {
    // Do something
  }
}
```

```
void f() {
  MONITORENTER o
  try {
    // Do something
  } finally {
    MONITOREXIT o
  }
}
```

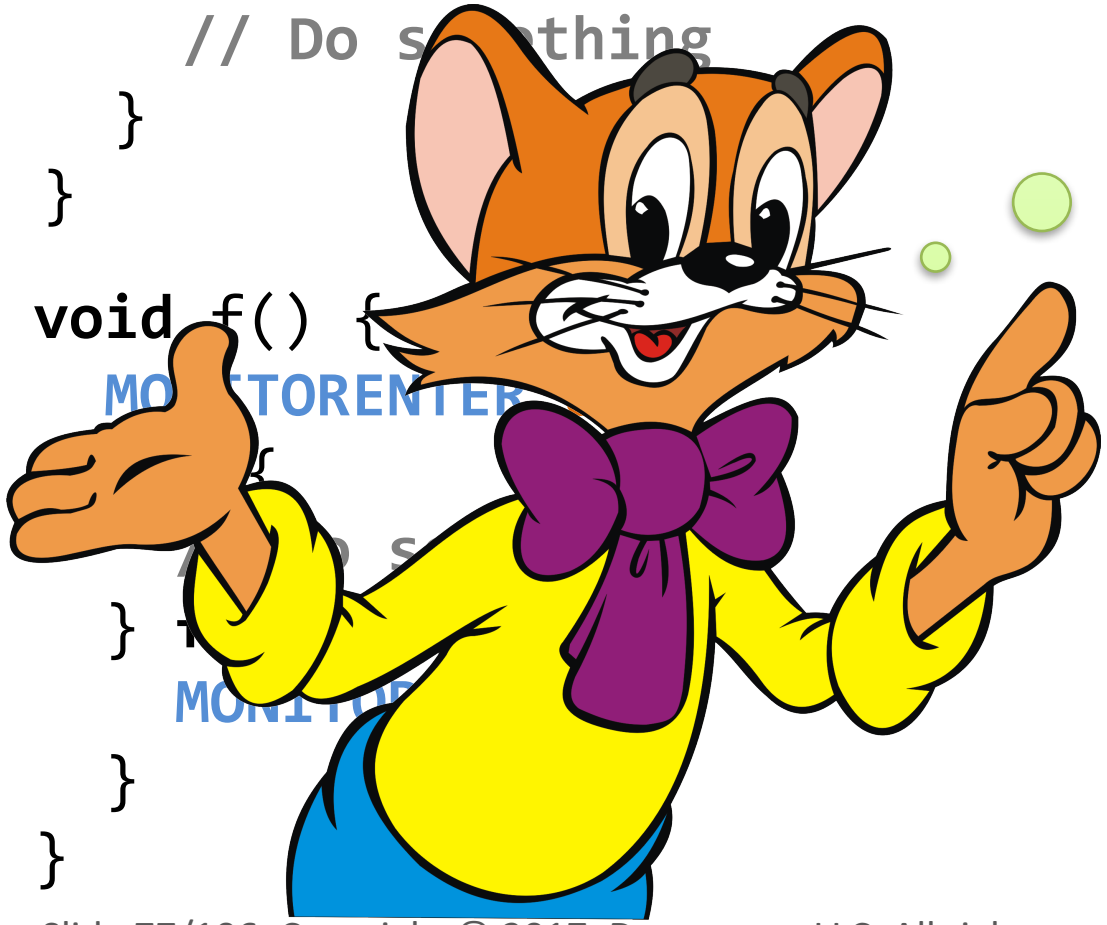
Lock acquisition analysis

Lock release analysis

```
void f() {
  MONITORENTER o
  try {
    onLockAcquire(o, 42);
    // Do something
  } finally {
    MONITOREXIT o
    onLockRelease(o);
  }
}
```

Synchronized block

```
void f() {
  synchronized (o) {
    // Do something
  }
}

void f() {
  MONITORENTER o
  try {
    // Do something
  } finally {
    MONITOREXIT o
  }
}
```



Not so simple!

```
void f() {
  MONITORENTER o
  try {
    onLockAcquire(o, 42);
    // Do something
  } finally {
    MONITOREXIT o
    onLockRelease(o);
  }
}
```


DEVEXPERTS

Synchronized block: backstage

```
void f() {
  synchronized (o) {
    // Do something
  }
}
```

```
Exception table:
  from   to  target type
     7    9      12  any
    12   15      12  any
```

```
void f();
Code:
try
  0: aload_0 // this
  1: getfield  #2 // monitor
  4: dup
  5: astore_1 // store monitor
  6: monitorenter
     // Do something
  7: aload_1 // monitor
  8: monitorexit
  9: goto  17 // skip finally
catch
 12: astore_2 // exception
 13: aload_1 // monitor
 14: monitorexit
 15: aload_2 // exception
 16: athrow
 17: return
```


DEVEXPERTS

Synchronized block: exception

WRONG

```
void f() {
  MONITORENTER o
  try {
    onLockAcquire(o, 42);
    // Do something
  } finally {
    MONITOREXIT o
    onLockRelease(o);
  }
}
```

CORRECT

```
void f() {
  MONITORENTER o
  try {
    onLockAcquire(o, 42);
    // Do something
  } finally {
    MONITOREXIT o
    try {
      onLockRelease(o);
    catch (Throwable t) {}
  }
}
```


DEVEXPERTS

DEVEXPERTS

Synchronized block: not a block

```
void f() {
  MONITORENTER o
  try {
     // Do something
  } finally {
    MONITOREXIT o
  }
}
```

```
void f() {
  MONITORENTER o
  try {
    onLockAcquire(o, 42);
  } catch (Throwable t) {
    MONITOREXIT o
    onLockRelease(o);
    throw t;
  }
  try {
     // Do something
     ...
```

Add try-catch

DEVEXPERTS

j.u.c.l.Lock: common usages

```
void f() {
  l.lock();
  try {
    // Do something
  } finally {
    l.unlock();
  }
}
```

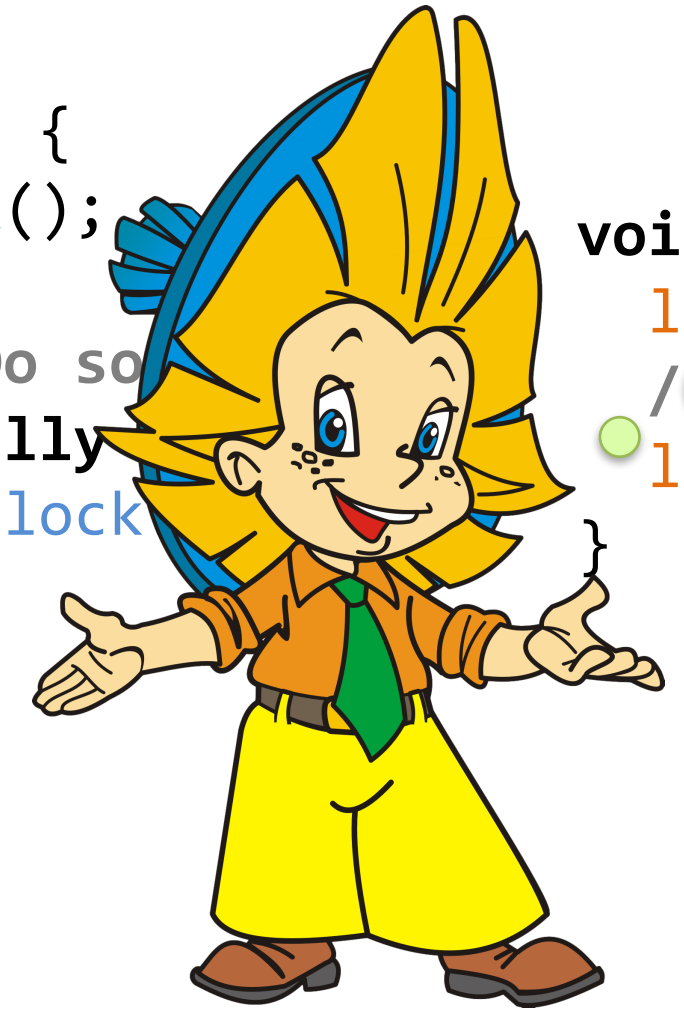
```
void g() {
  l.lock();
  // Do something
  l.unlock();
}
```

```
void h() {
  b = l.tryLock();
  // Do something
  l.unlock();
}
```


DEVEXPERTS

j.u.c.l.Lock: general usages

```
void f() {
  l.lock();
  try {
    // Do so
  } finally
    l.unlock
  }
}
```



```
void g() {
  l.loc    ;
  // Do something
  l.unlock();
}
```

```
    tryLock();
  // Do something
  l.unlock();
}
```

It looks like unpaired **MONITORENTER** and **MONITOREXIT**!

DEVEXPERTS

j.u.c.l.Lock: byte-code

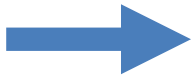
```
void g() {
  l.lock();
  // Do something
  l.unlock();
}
```

```
void g();
Code:
   0: aload_0 // this
   1: getfield  #3 // Lock l
   4: invokeinterface #4, 1 // lock()
      // Do something
   9: aload_0 // this
  10: getfield  #3 // Lock l
  13: invokeinterface #5, 1 // unlock()
  18: return
```


DEVEXPERTS

j.u.c.l.Lock: tryLock

```
void h() {
  b = l.tryLock();
  // Do something
  l.unlock();
}
```



```
void h() {
  b = l.tryLock();
  if (b) {
    try {
      onLockAcquire(l, 2);
    } catch (Throwable t) {
      l.unlock();
      onLockRelease(l);
      throw t;
    }
  }
  // Do something
  l.unlock();
  onLockRelease(l);
}
```


DEVEXPERTS

ClassTransformer

Code template:

```
public final byte[] transform(..., classFileBuffer) {
  ClassReader cr = new ClassReader(classFileBuffer);
  ClassWriter cw = new ClassWriter(...);
  ClassVisitor classTransformer = new MyTransformer(cw, ...);
  cr.accept(classTransformer, EXPAND_FRAMES);
  return cw.toByteArray();
}
```

Let’s try to run…

DEVEXPERTS

ClassTransformer: LinkageError

```
java.lang.LinkageError: loader (instance of
sun/misc/Launcher$AppClassLoader): attempted duplicate class definition
for name: "com/intellij/util/lang/UrlClassLoader"
  at java.lang.ClassLoader.defineClass1(Native Method)
  at java.lang.ClassLoader.defineClass(ClassLoader.java:763)
  at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
  at java.net.URLClassLoader.defineClass(URLClassLoader.java:467)
  at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
  at java.net.URLClassLoader$1.run(URLClassLoader.java:368)
  at java.net.URLClassLoader$1.run(URLClassLoader.java:362)
  at java.security.AccessController.doPrivileged(Native Method)
  at java.net.URLClassLoader.findClass(URLClassLoader.java:361)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
  ...
```


DEVEXPERTS

ClassTransformer: LinkageError

```
java.lang.LinkageError: loader (instance of
sun/misc/Launcher$Ar                                    uplicate class definition
for name: "com/int
  at java.lang.C                                        hod)
  at java.lang.Cl                                       r.java:763)
  at java.securit                                       ass(SecureClassLoader.java:142)
  at java.net.URLClass                                  ss(URLClassLoader.java:467)
  at java.net.URLClassLoader.access$100(URL             assLoade
  at java.net.URLClassLoader$1.run(URLClassLoa          r.java
  at java.net.URLClassLoader$1.run(URLClassLoader.java
  at java.security.AccessController.doPrivileged(N      ive M
  at java.net.URLClassLoader.findClass(URLClassLoad     1)
  at java.lang.ClassLoader.loadClass(ClassLoader.ja
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java
  at java.lang.ClassLoader.loadClass(ClassLoader.     a:357)
  ...
```

F***ing ASM!

Problem ClassWriter

```
protected String getCommonSuperClass(String type1,
                                     String type2) {
  Class<?> c1, c2;
  ClassLoader loader = getClass().getClassLoader();
  try {
    c1 = Class.forName(type1.replace('/', '.'), false, loader);
    c2 = Class.forName(type2.replace('/', '.'), false, loader);
  } catch (Exception e) {
    throw new RuntimeException(e);
  }
  if (c1.isAssignableFrom(c2))
    return type1;
  ...
```


DEVEXPERTS

Write it yourself!

```
@Override
protected String getCommonSuperClass(String type1,
                                     String type2) {
  ClassInfo c = ciCache.getOrBuildClassInfo(type1, loader);
  ClassInfo d = ciCache.getOrBuildClassInfo(type2, loader);
  if (c.isAssignableFrom(d, ciCache, loader))
    return type1;
  if (d.isAssignableFrom(c, ciCache, loader))
    return type2;
  ...
```

`ClassInfo` is available **BEFORE** transformation

DEVEXPERTS

Hooray! It works!

JAgent framework

- This problem is solved in JAgent
  - https://github.com/Devexperts/jagent
- Makes java agents writing more simple

DEVEXPERTS

Dependency conflict

- Several agents are used:

```
java -javaagent:agent1.jar -javaagent:agent2.jar
-jar my_app.jar
```

- Dependencies should be isolated

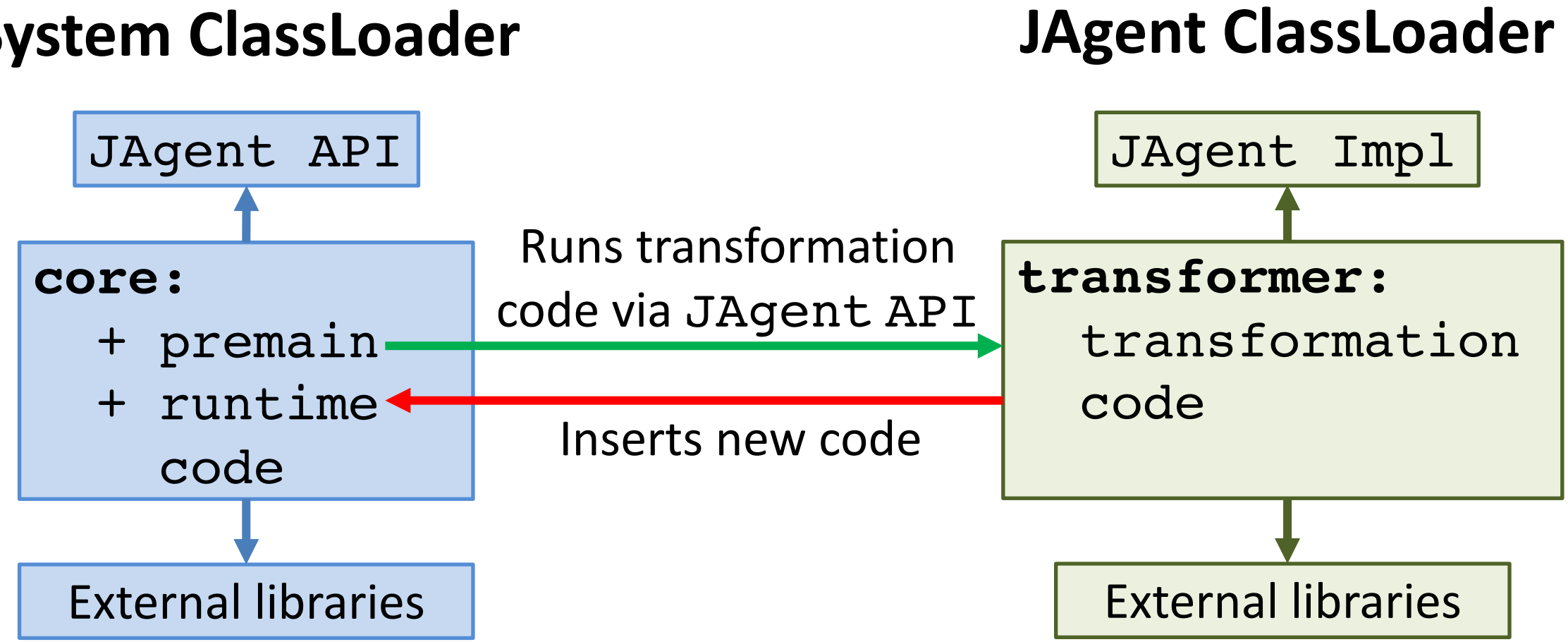
DEVEXPERTS

Isolation techniques

- Repackaging
  - `maven-shade-plugin` (or analogue)
  - Needs to know transitive dependencies and their structure
  - Sometimes doesn't work as well (reflection)
- Separate ClassLoader
  - Cannot be used for injected code

DEVEXPERTS

JAgent will help us



DEVEXPERTS

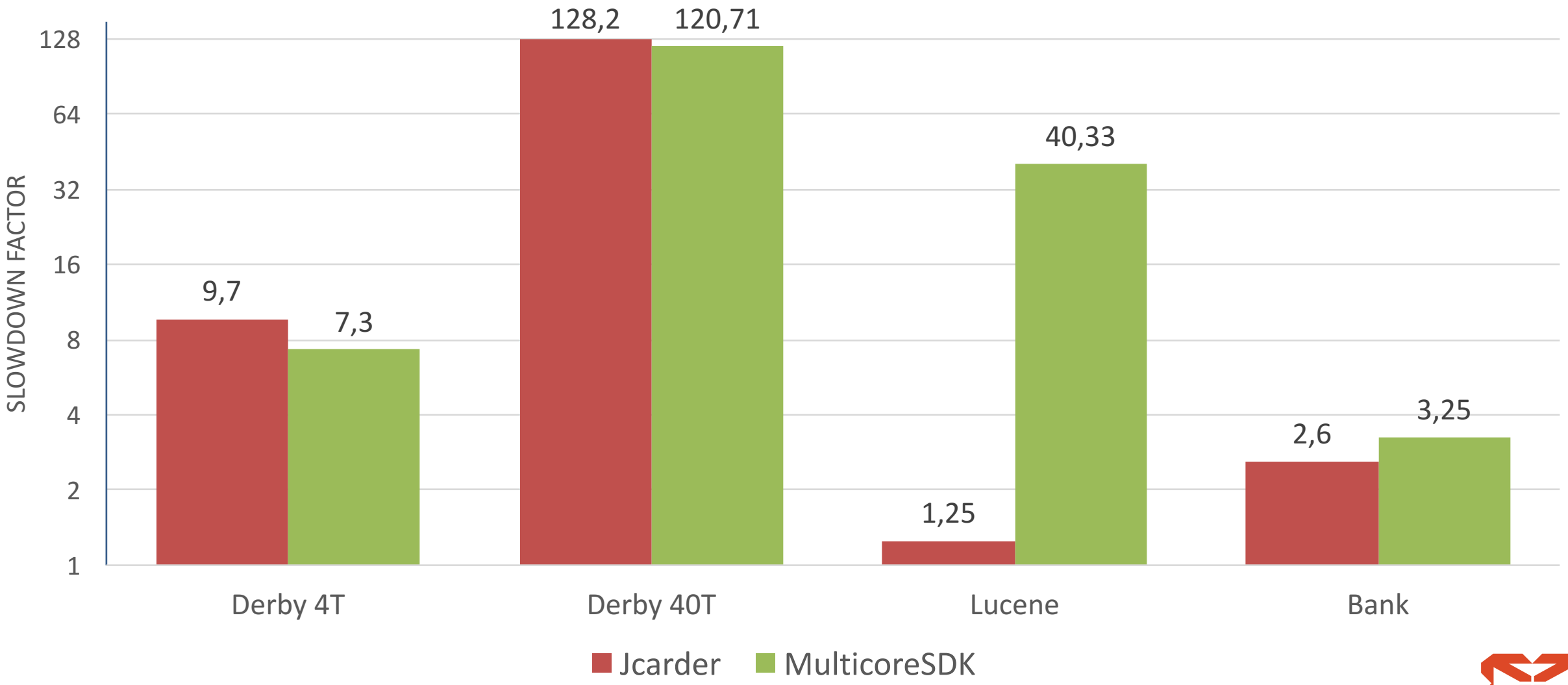
EVALUATION

Benchmarks

- SpecJVM2008: Apache Derby (4 and 40 threads)
- DaCapo: Apache Lucene
- DaCapo: Banking application
- …

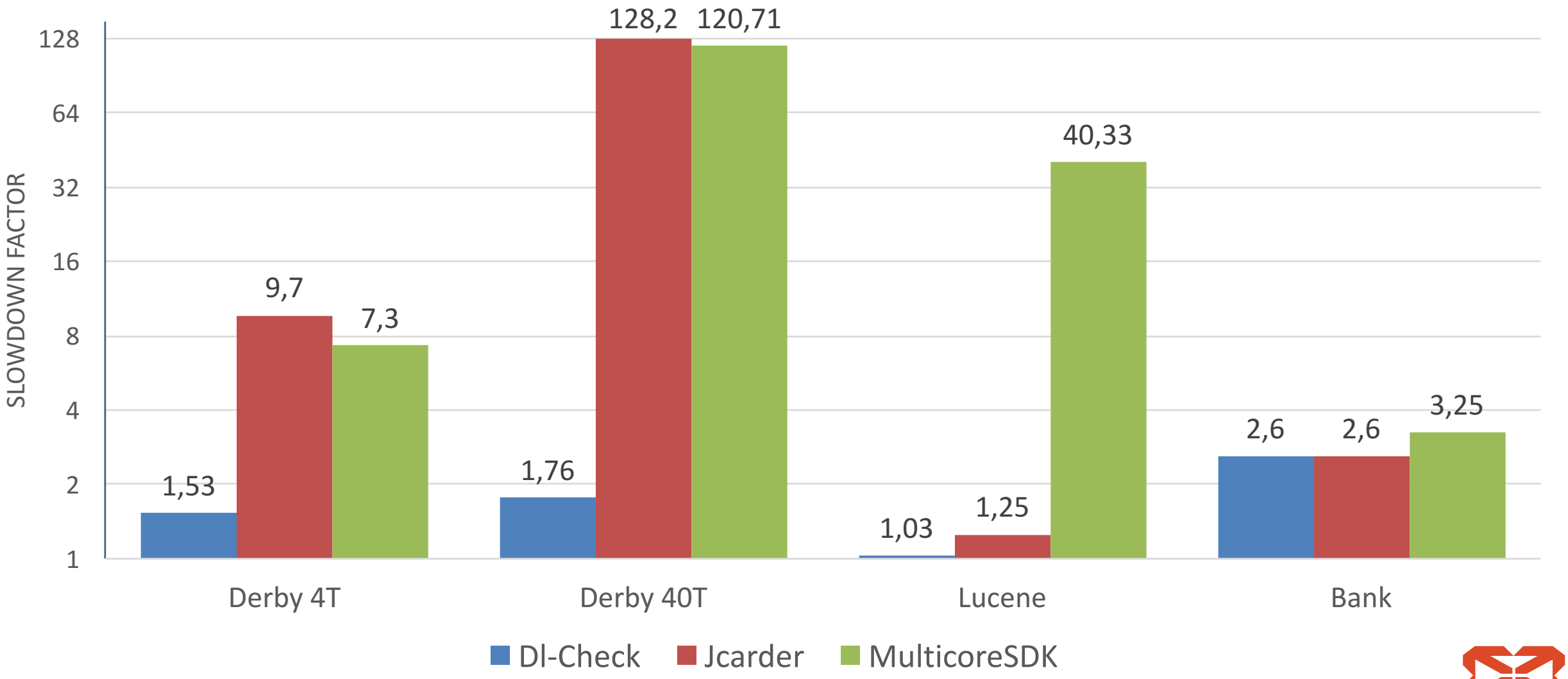
DEVEXPERTS

Performance impact



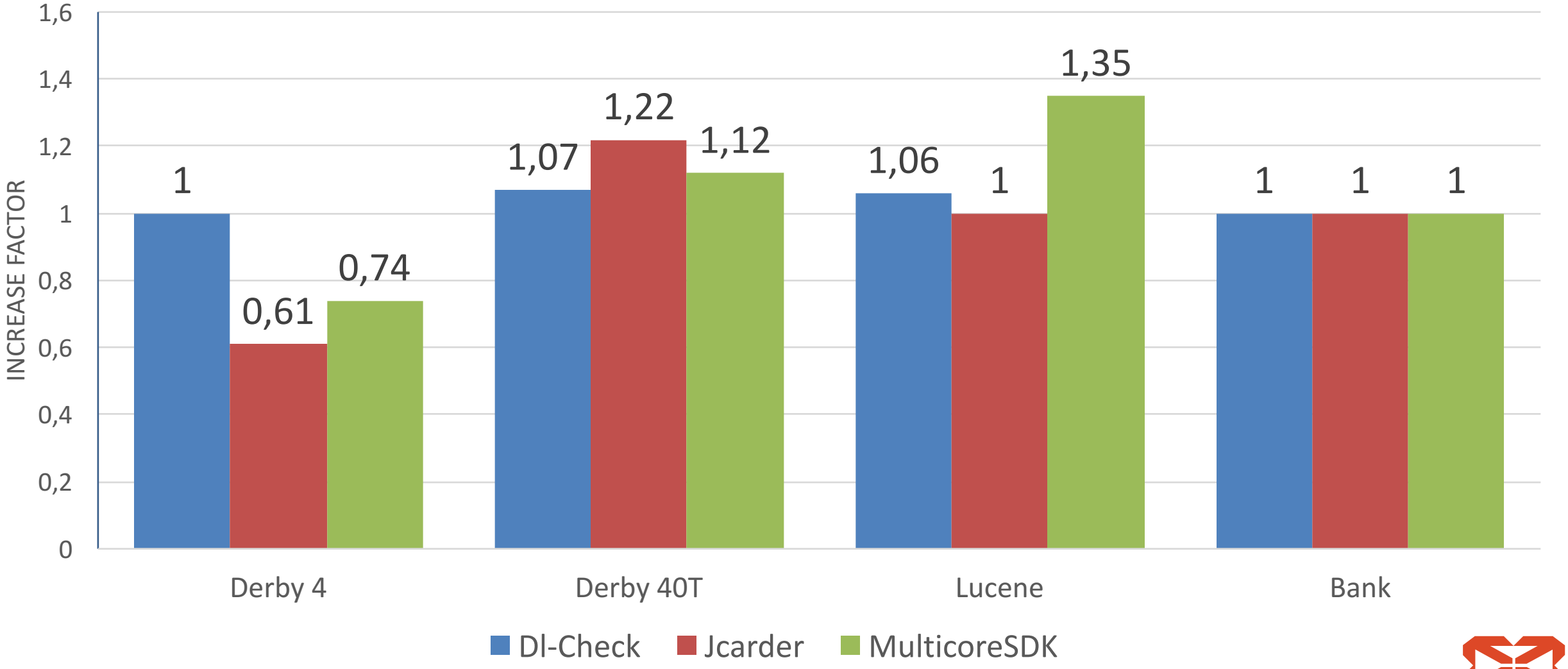
DEVEXPERTS

Performance impact



DEVEXPERTS

Memory usage impact



DEVEXPERTS

Real-life example

DEVEXPERTS

```
public void testClose() {
   cm.register(con);
      // lock cm
      // lock con for con.onRegister()
   assertEquals(1, cm.getConCount());
   con.close();
      // lock con
      // lock cm for cm.unregister()
   assertEquals(0, cm.getConCount());
}
```


DEVEXPERTS

```
==========================
!!! Potential deadlock !!!
==========================
### Cycle in lock graph: ###
Lock ConnectionImpl@d70c109 was acquired at:
      a.ConnectionImpl.onRegister(Sample.java:39)
      a.ConnectionImpl.close(Sample.java:44)
Lock ConnectionManager@2be94b0f was acquired at:
      a.ConnectionManager.getConCount(Sample.java:24)
      a.ConnectionManager.register(Sample.java:15)
      a.ConnectionManager.unregister(Sample.java:20)
### Current lock stack: ###
Lock ConnectionImpl@d70c109 was acquired at:
      a.ConnectionImpl.onRegister(Sample.java:39)
      a.ConnectionImpl.close(Sample.java:44)
### Current stacktrace: ###
      a.ConnectionManager.unregister(Sample.java:20)
      a.ConnectionImpl.close(Sample.java:45)
      a.FunctionalityTest.testClose(FunctionalityTest.java:22)
      ...
```


DEVEXPERTS

Evaluation in real projects

- Internal projects
- IntelliJ IDEA
- Kotlin
- …

DEVEXPERTS

Summary

- On-line algorithm for detecting potential deadlocks
- `Dl-Check` tool has been implemented
  - https://github.com/Devexperts/dlcheck
- Further work:
  - Contracts to describe lock acquisition rules

DEVEXPERTS

Thank you for attention!

Nikita Koval, Devexperts LLC

nkoval@devexperts.com
twitter.com/nkoval_