Type-Based Race Detection for Java
Programming With Threads

• Decompose program into pieces that can run in parallel
• Advantages
  - exploit multiple processors
  - threads make progress, even if others are blocked
Multithreaded Program Execution

Thread 1

\[ t_1 = \text{hits}; \]
\[ \text{hits} = t_1 + 1; \]
\[ \ldots \]

Thread 2

\[ t_2 = \text{hits}; \]
\[ \text{hits} = t_2 + 1; \]
\[ \ldots \]
A *race condition* occurs if two threads access a shared variable at the same time, and at least one of the accesses is a write.
Preventing Race Conditions Using Locks

- Lock can be held by at most one thread at a time
- Race condition prevention
  - associate a lock with each shared variable
  - acquire lock before accessing variable

Thread 1
```java
synchronized(lock) {
    int t1 = hits;
    hits = t1 + 1
}
```

Thread 2
```java
synchronized(lock) {
    int t2 = hits;
    hits = t2 + 1
}
```
Problem With Current Practice

• Locking discipline is not enforced
  - inadvertent programming errors cause races

• Race conditions are insidious bugs
  - non-deterministic, timing dependent
  - data corruption, crashes
  - difficult to detect, reproduce, eliminate

• Linux 2.4 log has 36 synchronization bug fixes
Reliable Multithreaded Software

• Correctness Problem
  - does program behaves correctly for all inputs and all interleavings?
  - very hard to ensure with testing

• Use static checkers
  - type systems target sequential programs
  - need type systems for multithreaded programs!
Use Type System to Ensure Race Freedom

- Static type system prevents race conditions

- Programmer specifies synchronization discipline
  - lock protecting each field
  - locks held on entry to each method

- Type checker checks synchronization discipline
  - checks field accessed only when lock held
  - checks for all inputs and all interleavings
Synchronized Bank Account

class Account {
    private int balance = 0;
    private void update(int x) {
        balance = x;
    }
    public void deposit(int n) {
        synchronized(this) {
            update(balance + n);
        }
    }
}

Thread 1
acct.deposit(100);

Thread 2
acct.deposit(100);
class Account {
  private int balance = 0 /*# guarded_by this */;
  private void update(int x) /*# requires this */ {
    balance = x;
  }
  public void deposit(int n) {
    synchronized(this) {
      update(balance + n);
    }
  }
}
Annotated Account

class Account {
    private int balance = 0           guarded_by this ;
    private void update(int x)        requires this    {
        balance = x;
    }
    public void deposit(int n) {
        synchronized(this) {
            update(balance + n);
        }
    }
}

15 Jan 2004 290G - Lecture 4a 11 / 27
Annotated Account

```java
class Account {
    private int balance = 0  // guarded_by this
    private void update(int x) // requires this
        { balance = x; }
    public void deposit(int n) {
        synchronized(this) {
            update(balance + n);
        }
    }
}
```

- **Lock names are constant expressions:**
  - `this`
  - `x.f` (if `x` is a constant expression and `f` a final field)
  - `java.lang.System.out` (static final field)
class Account {
    private int balance = 0 guarded_by this;
    private void update(int x) requires this {
        lockset is {this}
        balance = x; this ∈ lockset ? Yes
    }

    public void deposit(int n) {
        lockset is empty
        synchronized(this) {
            lockset is {this}
            update(balance + n); {this} ⊆ lockset ? Yes
        }
    }
}
Typing Judgment

\[ P; E; ls \vdash e : t \]

Program  Environment  Lock set
Typing Rules

• Thread creation

\[
P; E; \emptyset \vdash e : t \\
\hline
P; E; ls \vdash \text{fork } e : \text{int}
\]

• Lock acquisition

\[
P; E \vdash_{\text{final}} e_1 : c \\
\hline
P; E; ls \cup \{e_1\} \vdash e_2 : t \\
\hline
P; E; ls \vdash \text{synchronized } e_1 \text{ in } e_2 : t
\]

lock is constant
add to lock set
Handling Aliases Using Substitutions

```java
class Account {
    int balance = 0 guarded_by this;
    ...
}

final Account acct = ... ;
synchronized(acct) {
    acct.balance = 100;
    this ∈ lockset ? No!
    this[this := acct]
    lockset is { acct }
    this ∈ lockset ? Yes!
    acct ∈ lockset ? Yes!
}
```
Externally Synchronized Account

class Account<Object lock> {
    private int balance = 0 guarded_by lock;
    private void update(int x) requires lock {
        balance = x;
    }
    public void deposit(int n) requires lock {
        update(balance + n);
    }
}

final Object aLock = new Object();
Account<aLock> acct = new Account<aLock>();
synchronized(aLock) { acct.deposit(100); }  // ok

acct.deposit(100); // error
Soundness of the Type System

- Soundness Guarantee:
  - well-typed programs do not have race conditions

- Some good programs have "benign races"
  - allow program to escape type system

```java
class Account {
    private int balance guarded_by this;
    public Account(int n) {
        balance = n;  // # no_warn
    }
}
```
Race-Free Type System Features

- Guarded fields
- Lock sets
- Aliases
- Parameterized classes
- Escapes
- Dependant types
- Subtyping
- Thread local analysis
- Constant analysis
- Arrays, ...
# Validation of Race Condition Checker

<table>
<thead>
<tr>
<th>Program</th>
<th>Size (lines)</th>
<th>Number of annotations</th>
<th>Annotation time (hrs)</th>
<th>Races Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashtable</td>
<td>434</td>
<td>60</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>Vector</td>
<td>440</td>
<td>10</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>java.io</td>
<td>16,000</td>
<td>139</td>
<td>16.0</td>
<td>4</td>
</tr>
<tr>
<td>Ambit</td>
<td>4,500</td>
<td>38</td>
<td>4.0</td>
<td>4</td>
</tr>
<tr>
<td>WebL</td>
<td>20,000</td>
<td>358</td>
<td>12.0</td>
<td>5</td>
</tr>
</tbody>
</table>

15 Jan 2004
## Validation of Race Condition Checker

<table>
<thead>
<tr>
<th>Program</th>
<th>Size (lines)</th>
<th>Number of annotations</th>
<th>Annotation time (hrs)</th>
<th>Races Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashtable</td>
<td>434</td>
<td>60</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>Vector</td>
<td>440</td>
<td>10</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>java.io</td>
<td>16,000</td>
<td>139</td>
<td>16.0</td>
<td>4</td>
</tr>
<tr>
<td>Ambit</td>
<td>4,500</td>
<td>38</td>
<td>4.0</td>
<td>4</td>
</tr>
<tr>
<td>WebL</td>
<td>20,000</td>
<td>358</td>
<td>12.0</td>
<td>5</td>
</tr>
<tr>
<td>TLC</td>
<td>53,500</td>
<td>n/a</td>
<td>n/a</td>
<td>4</td>
</tr>
<tr>
<td>orange</td>
<td>28,000</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>red</td>
<td>450,000</td>
<td>n/a</td>
<td>n/a</td>
<td>~20</td>
</tr>
</tbody>
</table>
class Vector {
    Object elementData[] guarded_by this;
    int elementCount        guarded_by this;

    int lastIndexOf(Object elem) {  
        return lastIndexOf(elem, elementCount - 1);
    }  

    synchronized int lastIndexOf(Object elem, int n) {  
        for (int i = n ; i >= 0 ; i--)
            if (elem.equals(elementData[i])) return i;
        return -1;
    }  

    synchronized boolean remove(int index) { ... }  
    synchronized void trimToSize() { ... }
}
class Vector {
    Object elementData[] guarded_by this;
    int elementCount guarded_by this;

    int lastIndexOf(Object elem) {
        return lastIndexOf(elem, elementCount - 1);
    }

    synchronized int lastIndexOf(Object elem, int n) {
        for (int i = n ; i >= 0 ; i--)
            if (elem.equals(elementData[i])) return i;
        return -1;
    }

    synchronized boolean remove(int index) { ... }
    synchronized void trimToSize() { ... }
}
java.util.Vector

```java
class Vector {
    Object elementData[] guarded_by this;
    int elementCount guarded_by this;

    int lastIndexOf(Object elem) {
        return lastIndexOf(elem, elementCount - 1);
    }

    synchronized int lastIndexOf(Object elem, int n) {
        for (int i = n; i >= 0; i--)
            if (elem.equals(elementData[i])) return i;
        return -1;
    }

    synchronized boolean remove(int index) { ... }
    synchronized void trimToSize() { ... }
}
```

RACE

IndexOutOfBoundsException