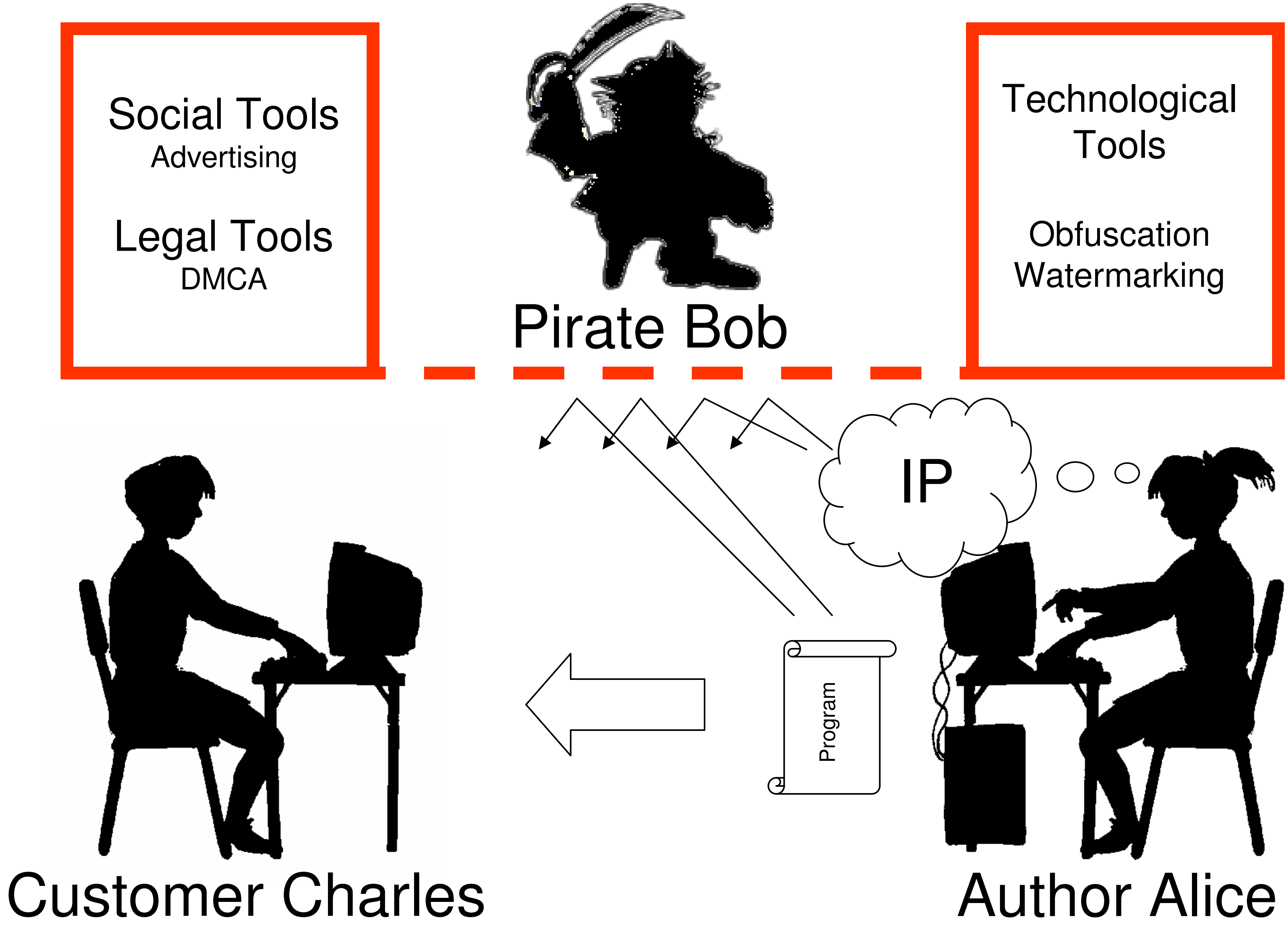
Threading Software Watermarks

University of Auckland
New Zealand

Jasvir Nagra
Clark Thomborson

Scenario


Social Tools
Advertising
Legal Tools
DMCA
Pirate Bob
Technological Tools
Obfuscation
Watermarking
IP
Program
Customer Charles
Author Alice

What is…

- Software Watermarking

  – embedding identifying information into a program

  – Program is semantically equivalent to the original
  – Embedded information can be extracted or detected
  – Recognition is potentially keyed on a secret key

What is…

- Dynamic Watermarking

  – embed information in the runtime behavior of a program

  – watermarked program must be run before the mark can be extracted
  – static analysis to determine runtime characteristics is hard

  – Only one other dynamic watermarking scheme exists: CT algorithm

Outline

How do you robustly embed information in a software program using threads?

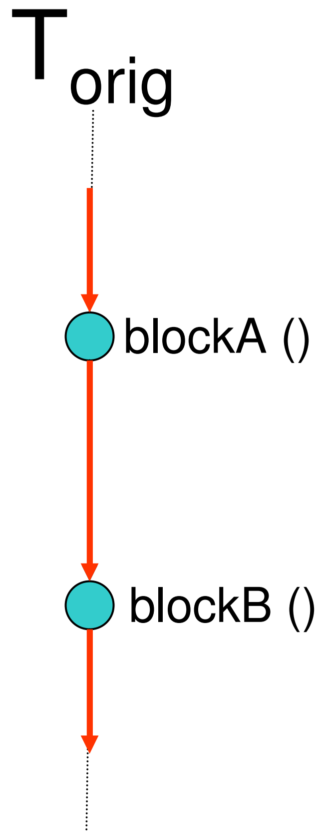
- We are interested in watermarks that are
  - robust
  - dynamic
- Outline
  - Using threads for encoding watermarks
  - Implementation in Java
  - Attacks against this scheme
  - Summary of notable features of our design

Why Threads?

- Ideally embedding a watermark should:
  - insert information
  - complicate analysis such that removing the watermark becomes difficult
- Static analysis of multithreaded programs is hard
  - Djikstra, Ousterhout, Collberg
- Collberg et al. suggest using threads for obfuscation
- We go further and show how to embed watermarks using threads

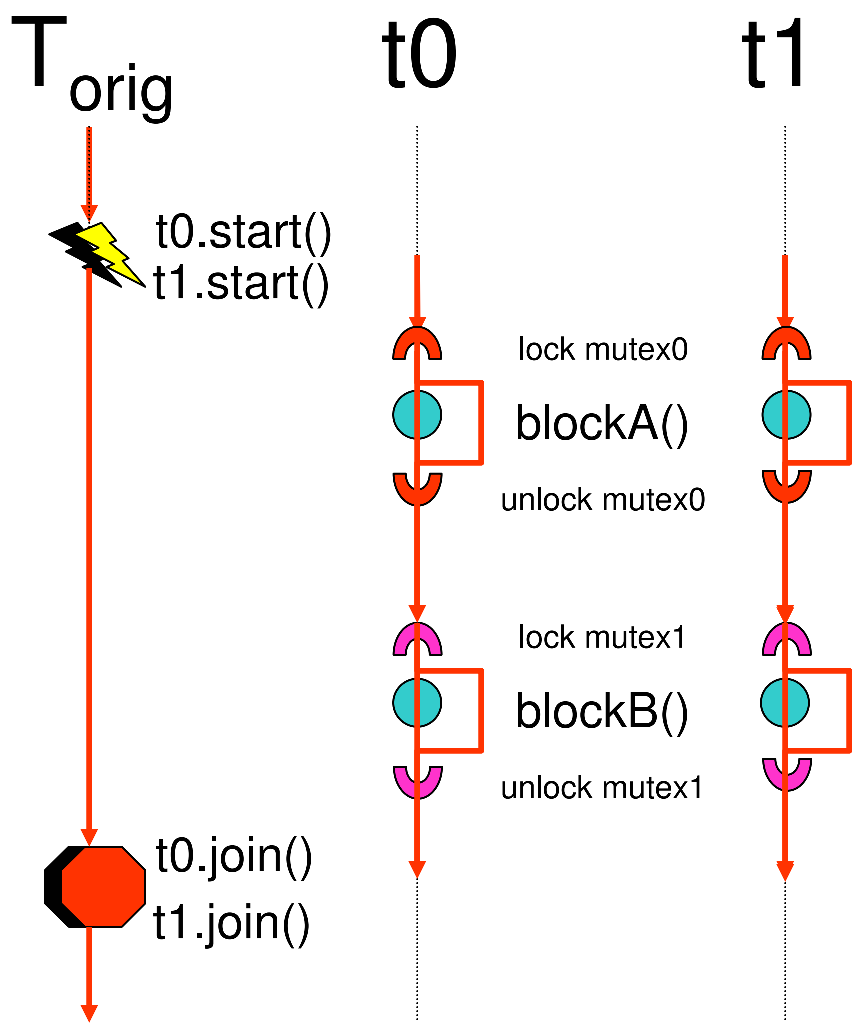
Encoding Information in Threads

```
public class Foo {

    public void run () {
        blockA();
        blockB();
    }

    :

    public static void main ( String args ) {
        Foo foo = new Foo();
        foo.run();
    }

}
```



Encoding Information in Threads

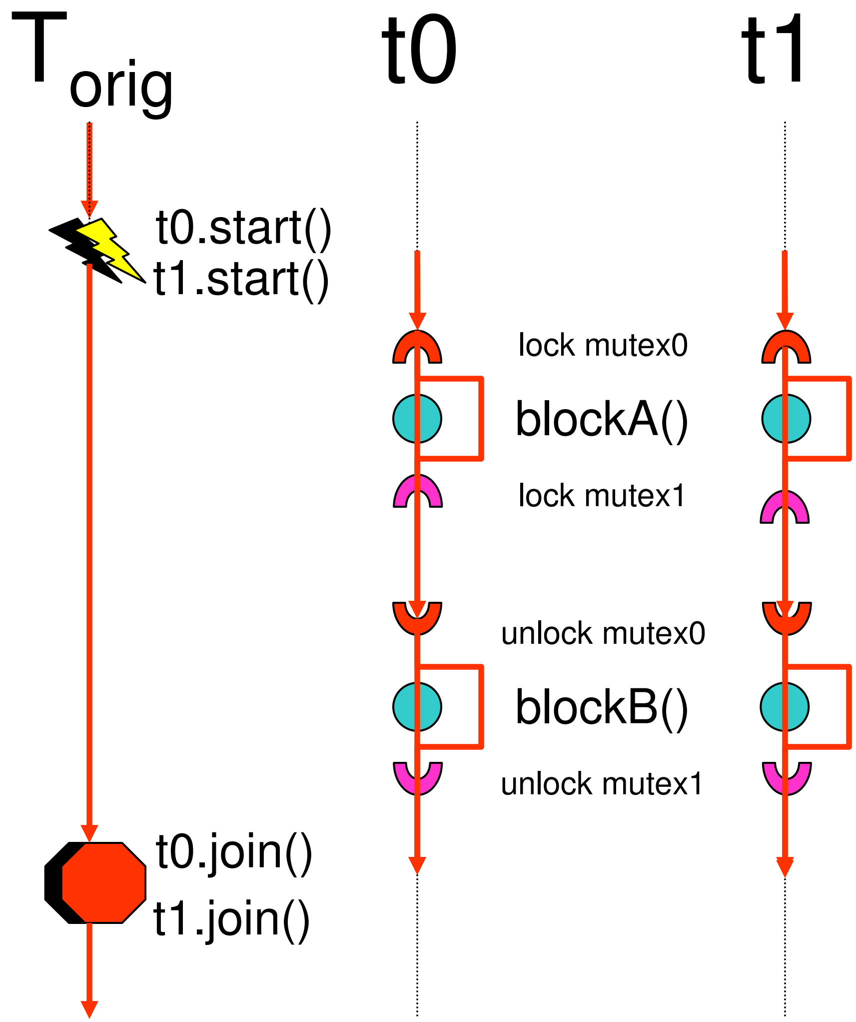
```
Thread t0 = new Thread () {
    public void run () {
            lock mutex0 ;
            if ( ! doneA ) {
                    blockA ();
                    doneA = true ;
            }
            unlock mutex0;
            lock mutex1;
            if ( ! doneB ) {
                    blockB ();
                    doneB = true ;
            }
            unlock mutex1;
    }
};
Thread t1 = new Thread ( t0 );
t1. start (); t0. start ();
t1. join (); t0. join ();
```



Four possible paths

Encoding Information in Threads

```
Thread t0 = new Thread () {
    public void run () {
        lock mutex0 ;
        if ( ! doneA ) {
            blockA ();
            doneA = true ;
        }
        lock mutex1;
        unlock mutex0;
        if ( ! doneB ) {
            blockB ();
            doneB = true ;
        }
        unlock mutex1;
    }
};
Thread t1 = new Thread ( t0 );
t1. start (); t0. start ();
t1. join (); t0. join ();
```



Two possible paths

Encoding Information in Threads

- Original program executes:
  - (t0, blockA), (t0, blockB)
  - (t0, blockA), (t1, blockB)
  - (t1, blockA), (t0, blockB)
  - (t1, blockA), (t1, blockB)

- Encoded program executes:
  - (t0, blockA), (t0, blockB)
  - (t1, blockA), (t1, blockB)
- Different encoded program:
  - (t0, blockA), (t1, blockB)
  - (t1, blockA), (t0, blockB)

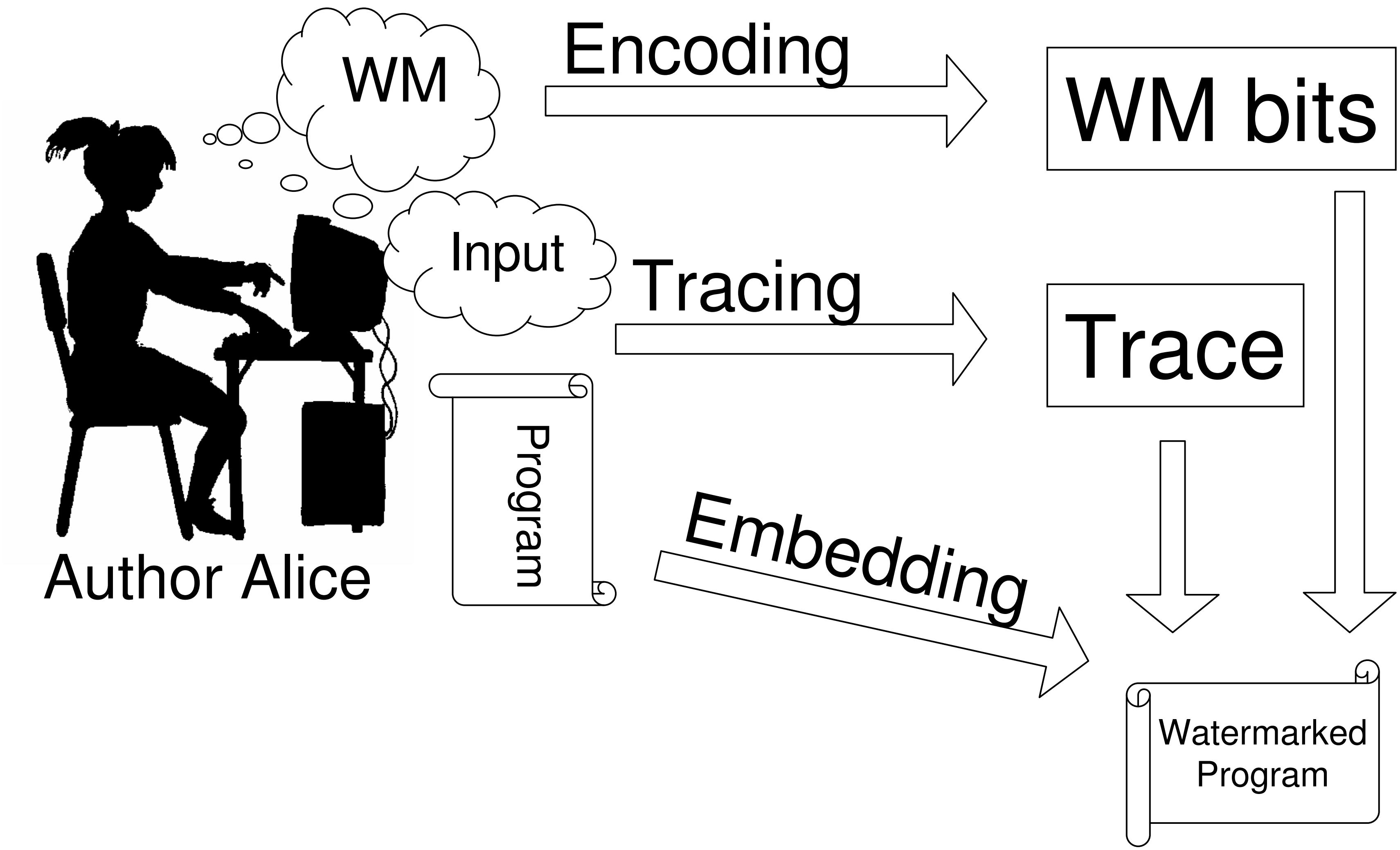
Can be used to encode a bit

Decompiling to Java

- Java requires well-nested monitors
  - at source level Java uses only a **synchronized** block
- Poorly nested monitors cannot be easily decompiled
  - **synchronized** blocks cannot capture these semantics
- No alteration of the JVM is needed
  - exploit semantic difference between Java language & JVM
  - dynamically all monitor enter and exit calls are matched
  - security guarantees made by Java are maintained
- One more impediment for the attacker

Thread-based Watermarking

- Overview



Encoding

"Copyright Alice, 2004"

Encoding function

436470797269…

Error correcting code

Bits to embed = 0100101001…

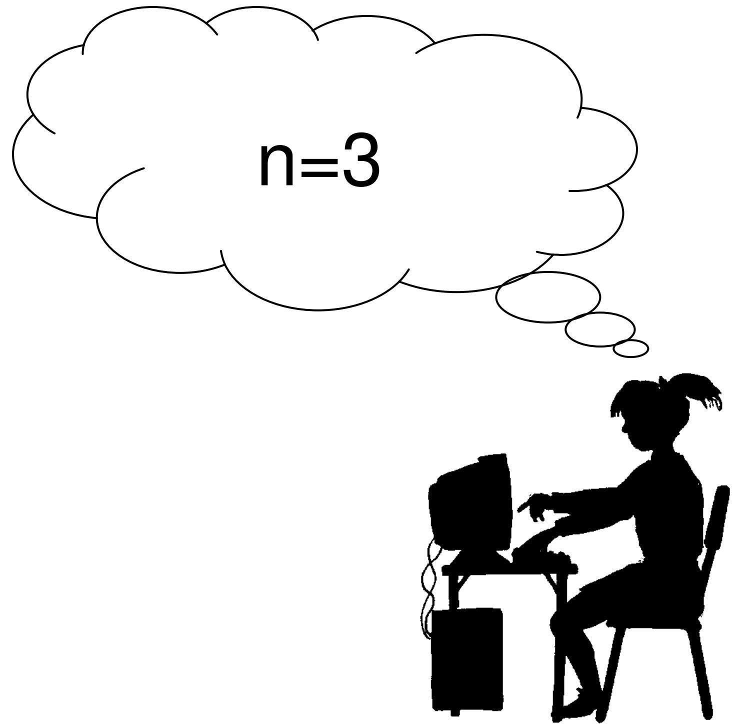
Tracing

```
public int foo ( int n ) {
    blockA;
    if ( blockB ) {
        blockC;
    } else {
        blockD;
        blockE;
    }
}
```

embed bit 0

embed bit 1

embed bit 1



- Alice selects a path through the program: A, B, D, E
- A subset of basic blocks that get executed on that path: A, D, E
- Embedding code in the basic blocks inserts the watermark
- For example Alice can embed “011” in A, D and E as shown

Embedding

- Divide each selected basic block into three pieces
- Create three new threads
- Execute the three pieces using the three threads
- Use locks to maintain semantic correctness
- Control which threads execute which piece

- Bit 0
  - (tA, piece1), (tB, piece2), (tC, piece3)

- Bit 1
  - (tA, piece1), (tB, piece2), (tA, piece3)

Detecting Thread Watermarks

(t3, mutex0)	(t1, mutex0)
(t3, mutex1)	(t3, mutexOrig)
(t2, mutex0)	(t3, mutex0)
⋮	(t2, mutex1)

- Annotate the program for tracing
- Run the program with secret input
- Decode the sequence of threads and locks found

Decoding Thread Watermarks

- Patterns of locks

  - Bit 0
    - tA, tA, tB, tC, tA, tA, tB
  - Bit 1
    - tA, tA, tB, tC, tC, tA, tB

Pattern Matching Attacks

- To keep the recognition dynamic, we have to prevent static pattern matching attacks distinguishing between bit0 and bit1

Bit 0

```
if ( doneC || doneD ) {
    ⋮
    monitorexit mutex1;
    ⋮
    monitorexit mutex0;
    ⋮
}
```

Bit 1

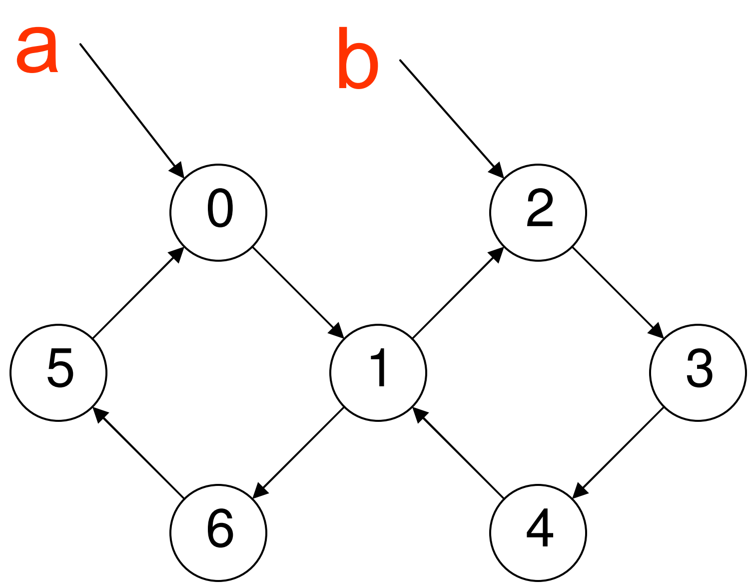
```
if ( ! doneD ) {
    ⋮
    monitorexit mutex0;
    ⋮
    monitorexit mutex1;
    ⋮
}
```

Pattern Matching Defense

- Static analysis will discover:
  - **monitorexit** takes different operands
  - predicates are different
- Merging predicates
  - use opaque predicates to collapse predicates
- Merging operands
  - operands to **monitorexit** in JVM appear on the stack
  - can obscure stack arguments
    - pointer aliasing

Opaque Predicates

- Opaque predicate is a
  - Non obvious tautology
  - Boolean expression
  - Value known to watermarker at watermarking time
  - Difficult for the attacker to deduce



Opaque Predicate Merge

- merge different predicates into a single statically indistinguishable predicate

- Bit 0
  (((doneC || doneD) && opaqueTrue) ||
  (!doneD && opaqueFalse))

- Bit 1
  (((doneC || doneD) && opaqueFalse) ||
  (!doneD && opaqueTrue))

Pattern Matching Defense

- Static analysis will discover:
  - predicates are different
  - **monitorexit** takes different operands
- Merging predicates
  - use opaque predicates to collapse predicates
- Merging operands
  - operands to **monitorexit** in JVM appear on the stack
  - can obscure stack arguments
    - pointer aliasing

Statistics

- Initial experiments using sample Java programs indicate:
  - Significant slowdown factor of ~8 on embedding a 48-bit watermark in a tightly optimized benchmark program without any I/O.
  - More modest slowdown factor <2 on GUI programs with a lot of user I/O
  - Achieved by avoiding tight loops and hotspots

  - Embedding a 48-bit watermark → 60kB increase in size
  - Size increase approx. linear with number of bits inserted

Evaluation

- Obfuscation Attacks
  - renaming attacks
  - block reordering
  - method inlining/outlining
- Decompilation/Recompilation Attacks
  - only well-nested monitors can be expressed using synchronized blocks at Java source level
  - current decompilers fail to decompile watermarked programs
  - decompilation is possible in theory
    - Dava emulates these locks using a library
- Additive Attacks
  - insert additional thread switches into the program
  - inserts additional bits into the decoded bit string

Conclusion

- Problem:

How can we use threads to embed information in a program?

- Solution
  - Encode the watermark as a bit string
  - Embed the bit string
    - locks control which threads execute selected basic blocks
  - Detect watermark
    - trace the order in which locks get acquired
- Stealth – prevent static analysis
  - Use pointer aliasing to hide which locks are used; and
  - Use opaque predicates to merge different predicates

Questions

- Questions?