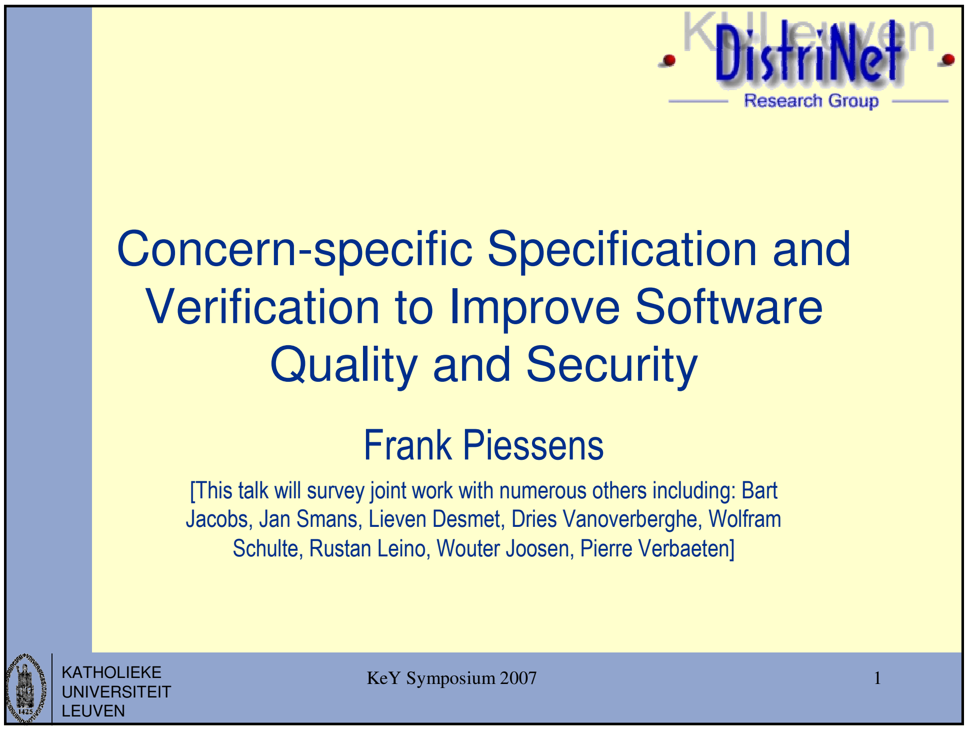


DistriNet Research Group

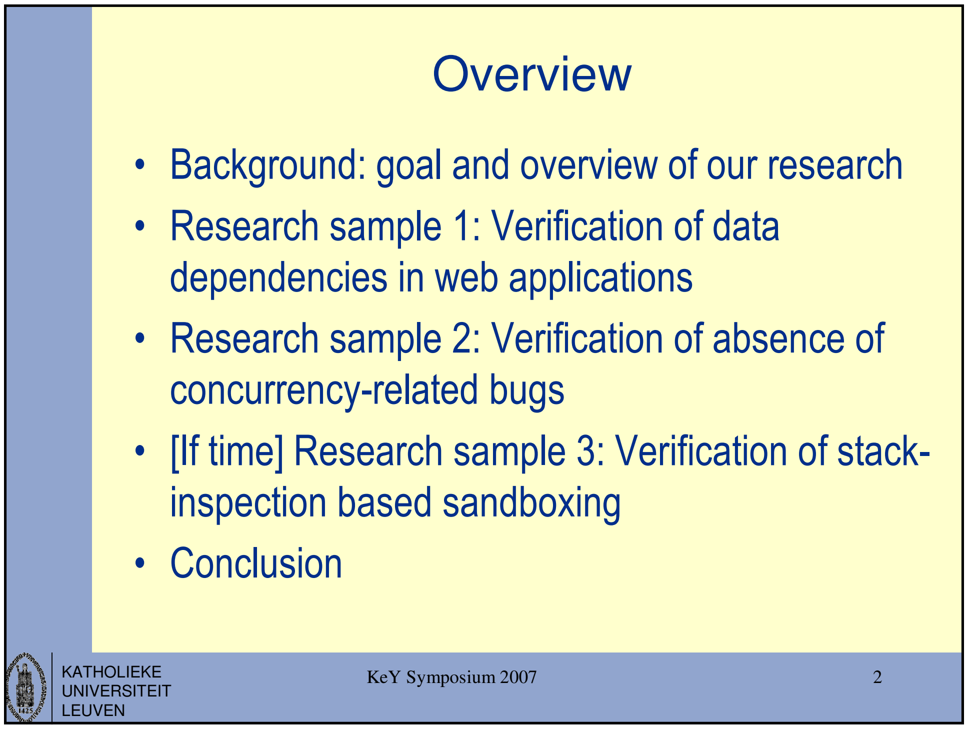
Concern-specific Specification and Verification to Improve Software Quality and Security

Frank Piessens

[This talk will survey joint work with numerous others including: Bart Jacobs, Jan Smans, Lieven Desmet, Dries Vanoverberghe, Wolfram Schulte, Rustan Leino, Wouter Joosen, Pierre Verbaeten]

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007 1

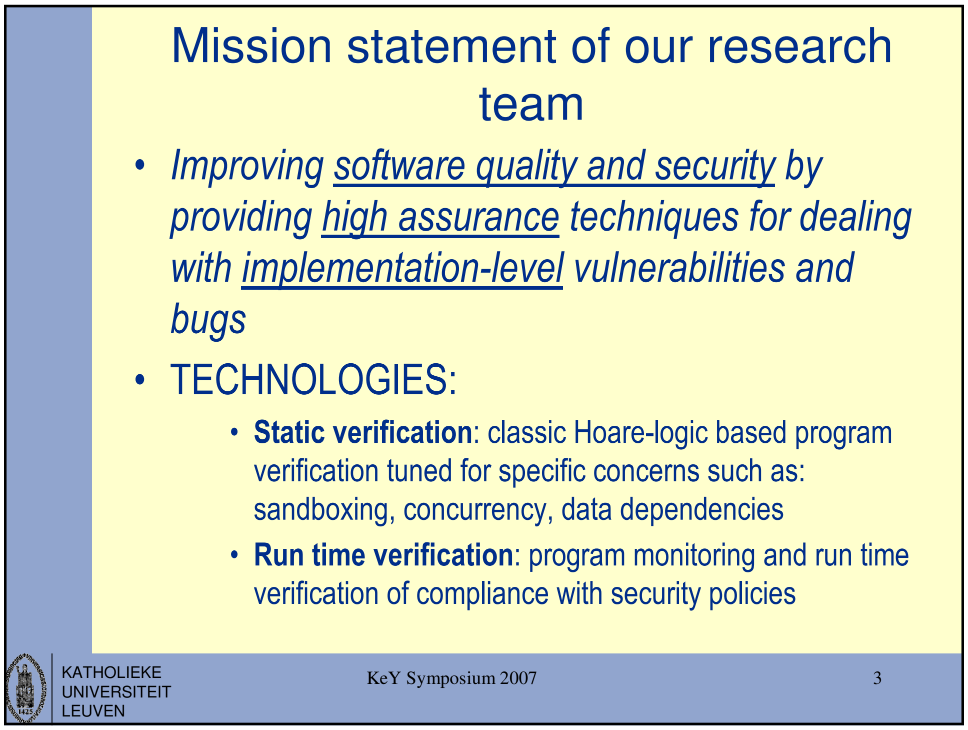


Overview

- Background: goal and overview of our research
- Research sample 1: Verification of data dependencies in web applications
- Research sample 2: Verification of absence of concurrency-related bugs
- [If time] Research sample 3: Verification of stack-inspection based sandboxing
- Conclusion

KATHOLIEKE UNIVERSITEIT LEUVEN

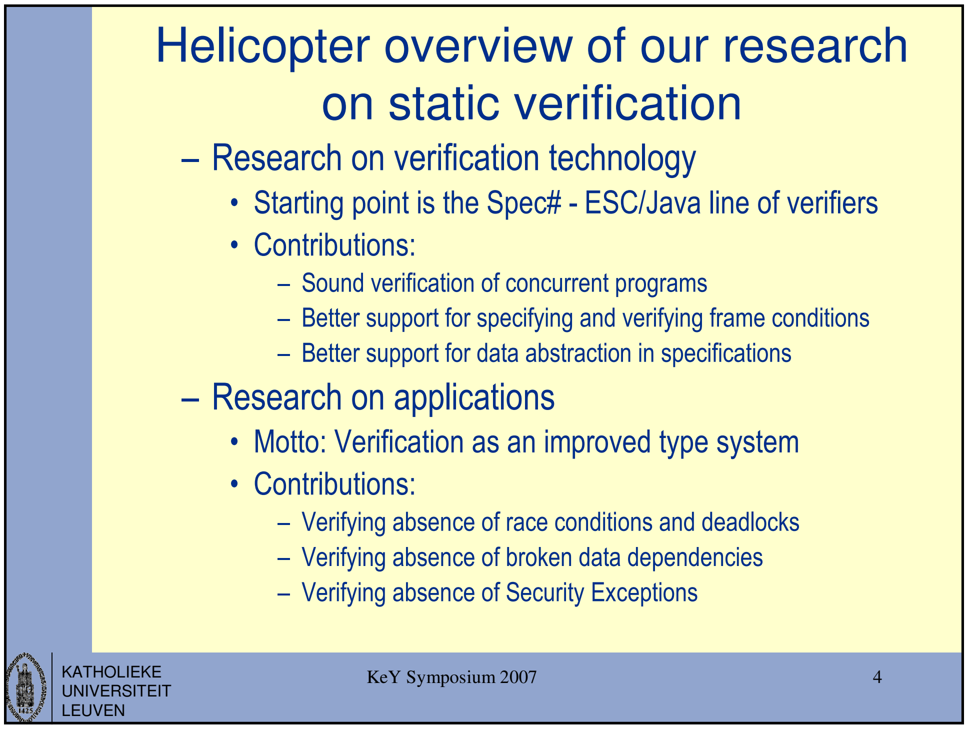
KeY Symposium 2007 2



Mission statement of our research team

- *Improving* <u>*software quality and security*</u> *by providing* <u>*high assurance*</u> *techniques for dealing with* <u>*implementation-level*</u> *vulnerabilities and bugs*
- TECHNOLOGIES:
  - **Static verification**: classic Hoare-logic based program verification tuned for specific concerns such as: sandboxing, concurrency, data dependencies
  - **Run time verification**: program monitoring and run time verification of compliance with security policies

KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 3

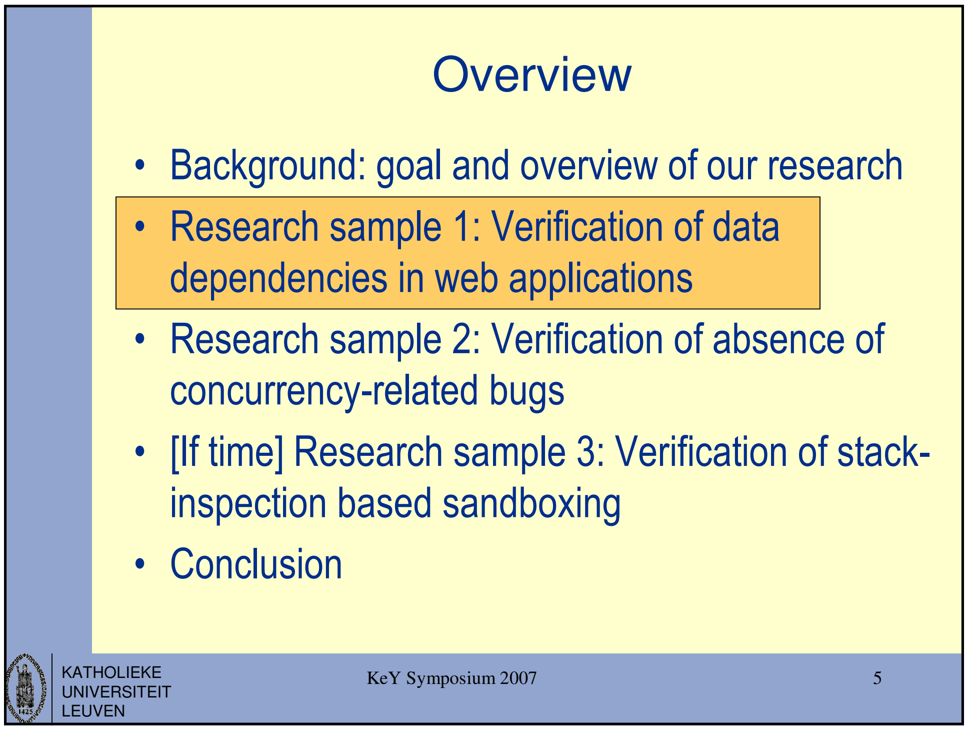


Helicopter overview of our research on static verification

– Research on verification technology
  - Starting point is the Spec# - ESC/Java line of verifiers
  - Contributions:
    – Sound verification of concurrent programs
    – Better support for specifying and verifying frame conditions
    – Better support for data abstraction in specifications

– Research on applications
  - Motto: Verification as an improved type system
  - Contributions:
    – Verifying absence of race conditions and deadlocks
    – Verifying absence of broken data dependencies
    – Verifying absence of Security Exceptions

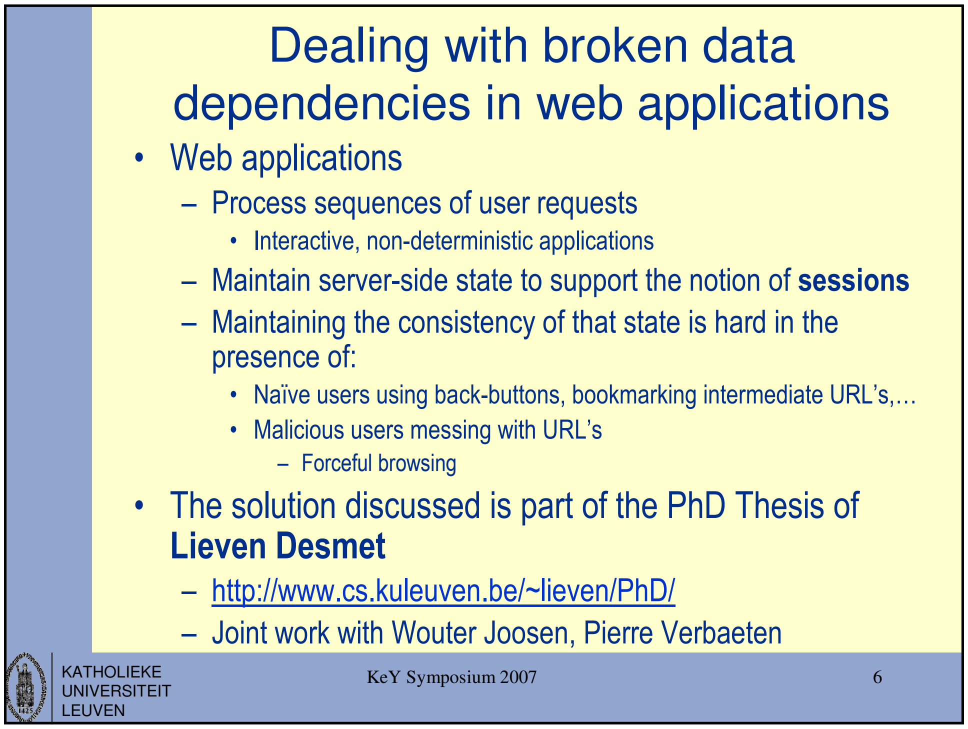
KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 4



Overview

- Background: goal and overview of our research
- Research sample 1: Verification of data dependencies in web applications
- Research sample 2: Verification of absence of concurrency-related bugs
- [If time] Research sample 3: Verification of stack-inspection based sandboxing
- Conclusion

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 5



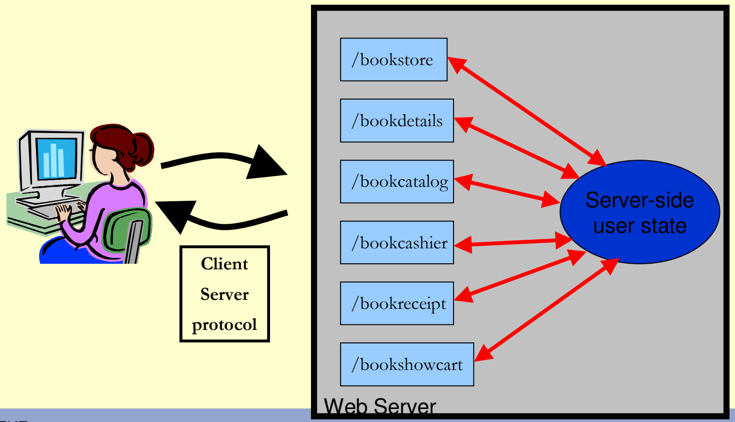
Dealing with broken data dependencies in web applications

- Web applications
  - Process sequences of user requests
    - Interactive, non-deterministic applications
  - Maintain server-side state to support the notion of **sessions**
  - Maintaining the consistency of that state is hard in the presence of:
    - Naïve users using back-buttons, bookmarking intermediate URL's,…
    - Malicious users messing with URL's
      - Forceful browsing
- The solution discussed is part of the PhD Thesis of **Lieven Desmet**
  - http://www.cs.kuleuven.be/~lieven/PhD/
  - Joint work with Wouter Joosen, Pierre Verbaeten

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 6

Duke's BookStore application

- E-commerce site bundled with the J2EE 1.4 tutorial
- Reactive client/server interaction



KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 7

Shared data interactions

- Session repository with 3 data items:
  - messages (*ResourceBundle*)
  - cart (*ShoppingCart*)
  - currency (*Currency*)

BookDetailsServlet:
ResourceBundle messages (read)
Currency currency (cond. def. read/write)

BookStoreServlet :
ResourceBundle messages (def. read/write)

ReceiptServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)

OrderFilter:
ShoppingCart cart (read)
Currency currency (read)

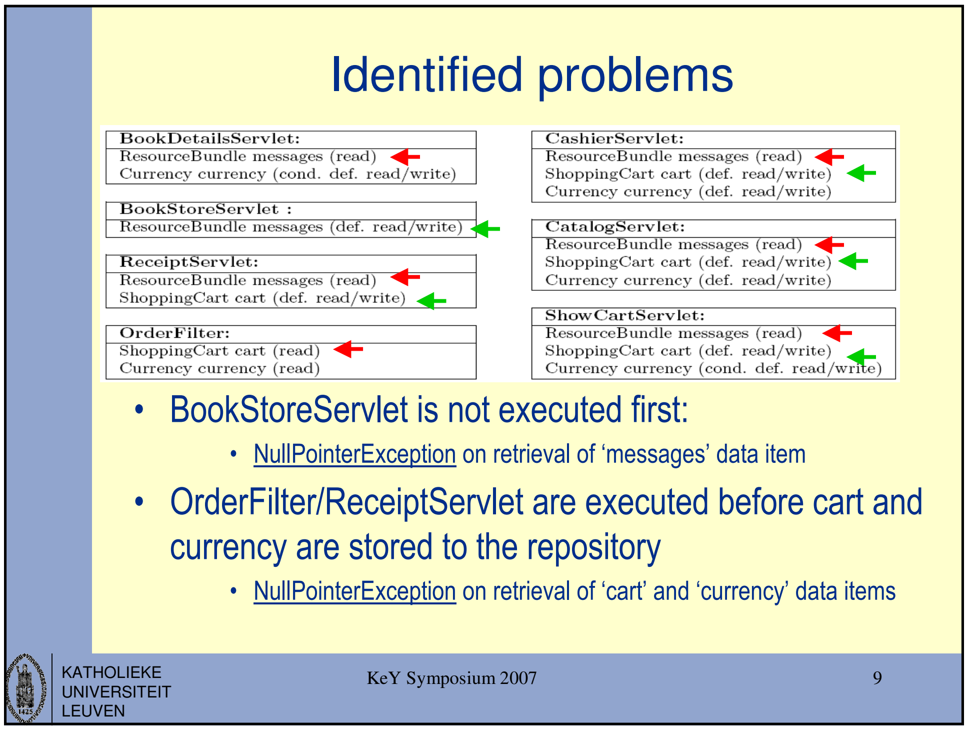
CashierServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (def. read/write)

CatalogServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (def. read/write)

ShowCartServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (cond. def. read/write)

  - read
  - def. read/write
  - cond. def. read/write

KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 8



Identified problems

BookDetailsServlet:
ResourceBundle messages (read)
Currency currency (cond. def. read/write)

BookStoreServlet :
ResourceBundle messages (def. read/write)

ReceiptServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)

OrderFilter:
ShoppingCart cart (read)
Currency currency (read)

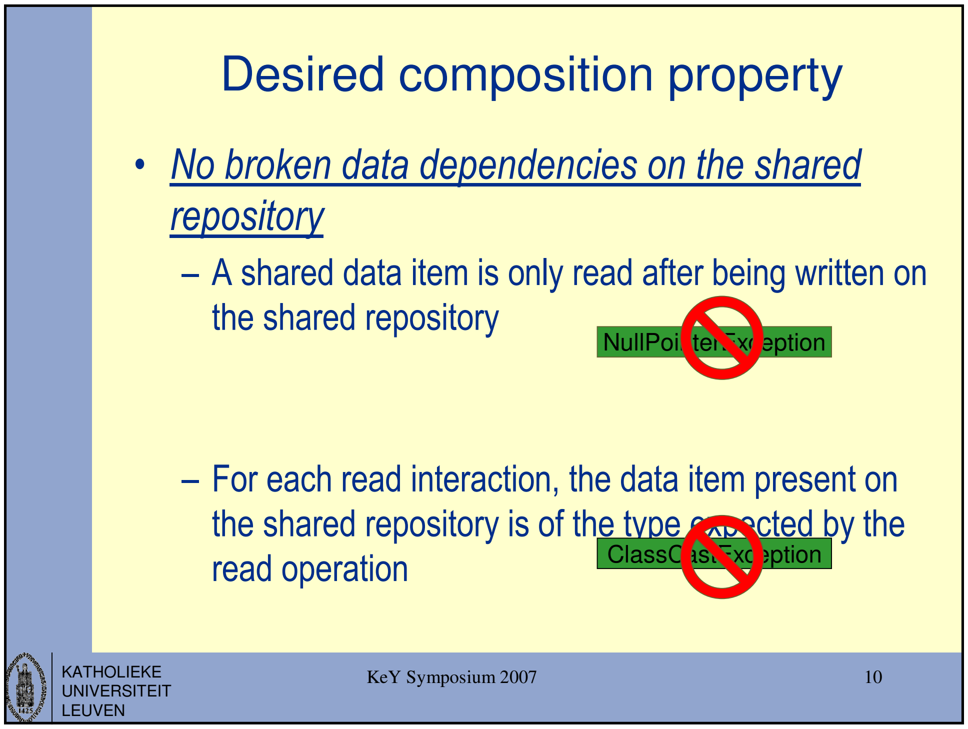
CashierServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (def. read/write)

CatalogServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (def. read/write)

ShowCartServlet:
ResourceBundle messages (read)
ShoppingCart cart (def. read/write)
Currency currency (cond. def. read/write)

- BookStoreServlet is not executed first:
  - NullPointerException on retrieval of 'messages' data item
- OrderFilter/ReceiptServlet are executed before cart and currency are stored to the repository
  - NullPointerException on retrieval of 'cart' and 'currency' data items

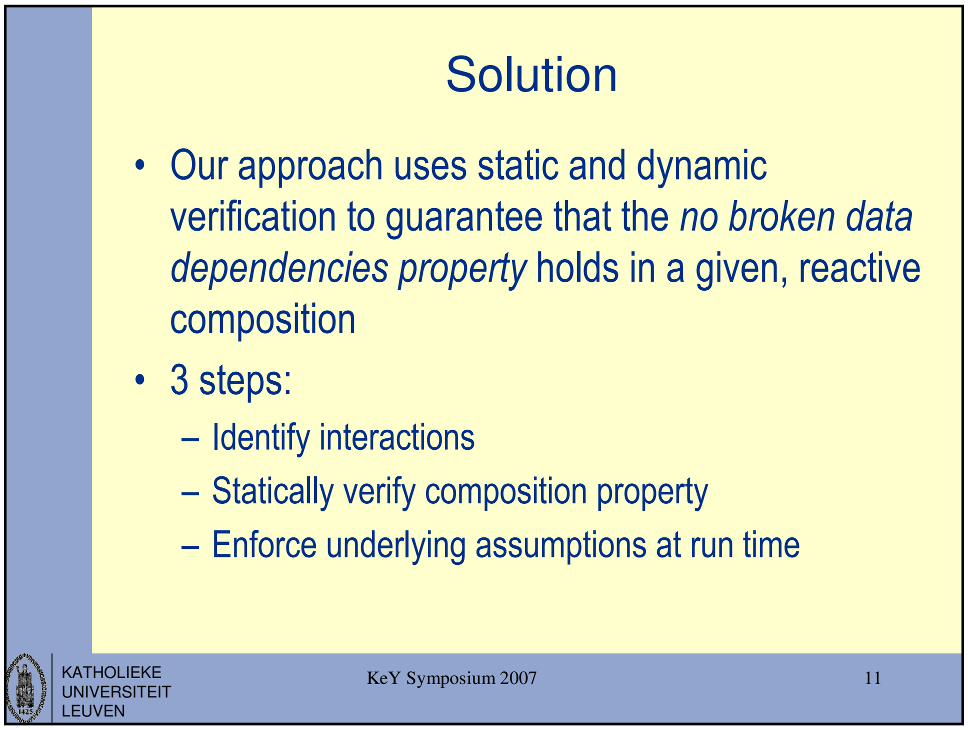
KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 9



Desired composition property

- *No broken data dependencies on the shared repository*
  - A shared data item is only read after being written on the shared repository
  NullPointerException
  - For each read interaction, the data item present on the shared repository is of the type expected by the read operation
  ClassCastException

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 10

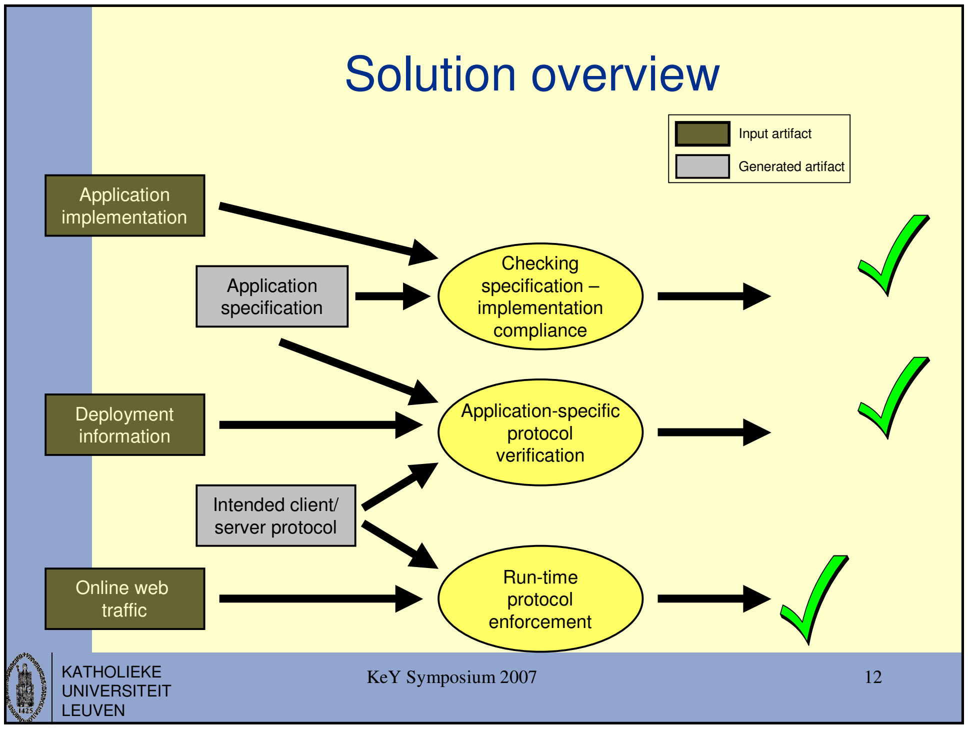


Solution

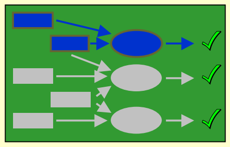
- Our approach uses static and dynamic verification to guarantee that the *no broken data dependencies property* holds in a given, reactive composition
- 3 steps:
  - Identify interactions
  - Statically verify composition property
  - Enforce underlying assumptions at run time

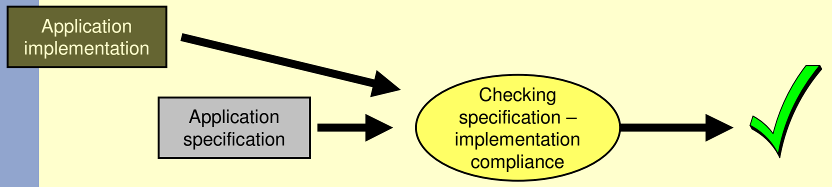
KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 11

Solution overview


Input artifact
Generated artifact
Application implementation
Application specification
Checking specification – implementation compliance
Deployment information
Application-specific protocol verification
Intended client/ server protocol
Online web traffic
Run-time protocol enforcement
KATHOLIEKE UNIVERSITEIT LEUVEN
KeY Symposium 2007
12

Step 1

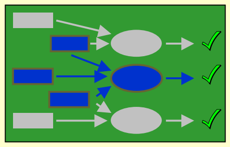


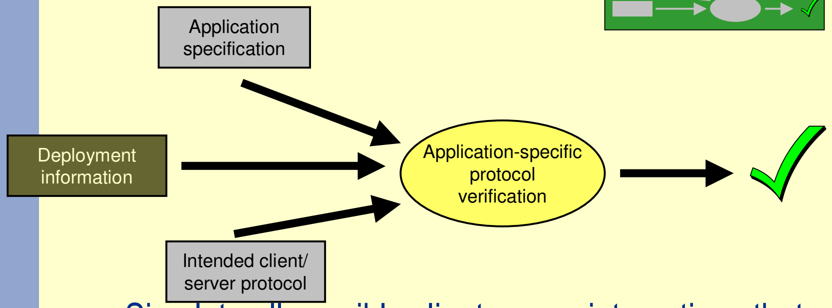

Application implementation
Application specification
Checking specification – implementation compliance

- Component contracts specify interactions with the shared repository:

```
//spec: reads {ResourceBundle messages, Nullable<ShoppingCart>cart,
                                          Nullable<Currency> currency} from session;
//spec: writes {cart == null => ShoppingCart cart} on session;
//spec: possible writes {currency == null => Currency currency} on session;
```

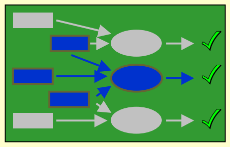
Step 2

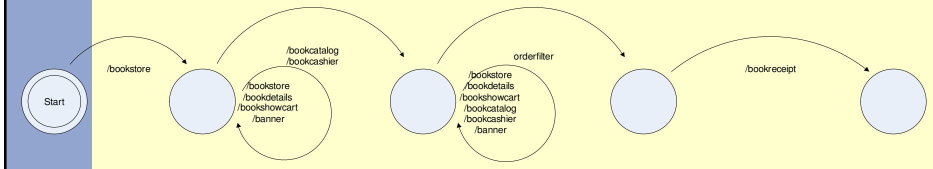



Application specification
Deployment information
Intended client/ server protocol
Application-specific protocol verification

- Simulate all possible client-server interactions that comply to the intended client/server protocol
- Use static verification to formally guarantee that the *no broken data dependency property* is not violated

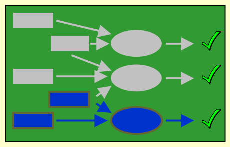
Intended client/server protocol

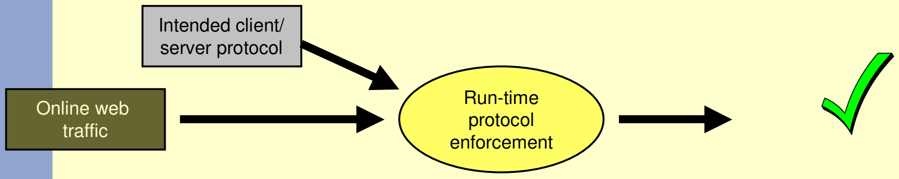




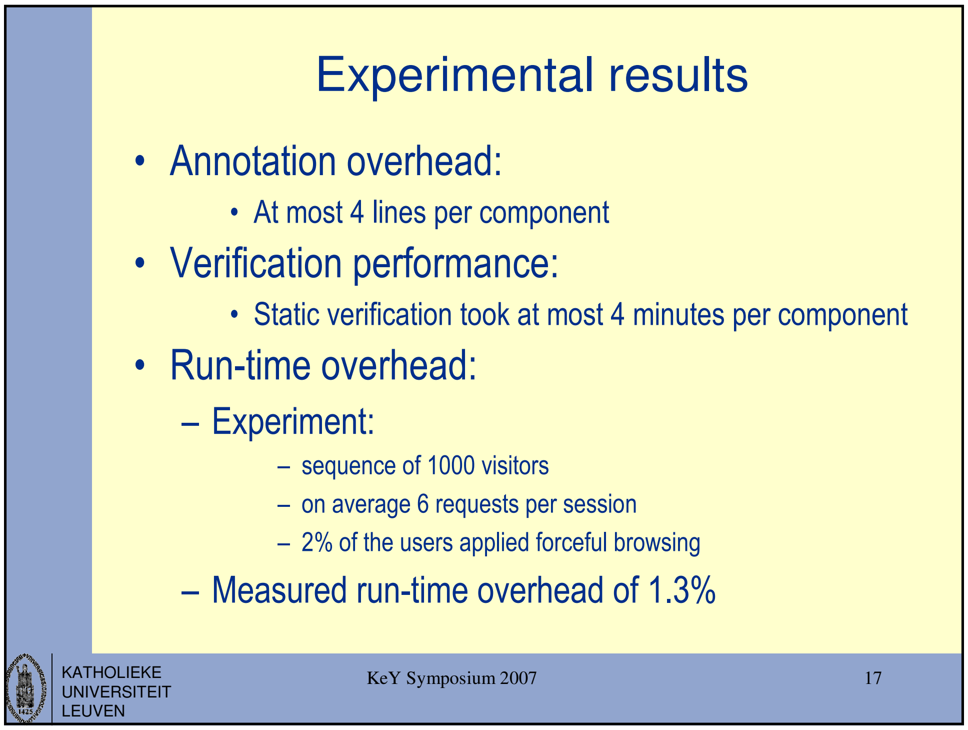
```
PROTOCOL := /bookstore + SERVLET A  + RECEIPT
RECEIPT := ( SERVLET B + SERVLET  + /orderfilter + /bookreceipt ) | nil
SERVLET := SERVLET A | SERVLET B
SERVLET A := /bookstore | /bookdetails | /bookshowcart | /banner | nil
SERVLET B := /bookcatalog | /bookcashier
```

Step 3




Intended client/ server protocol
Online web traffic
Run-time protocol enforcement

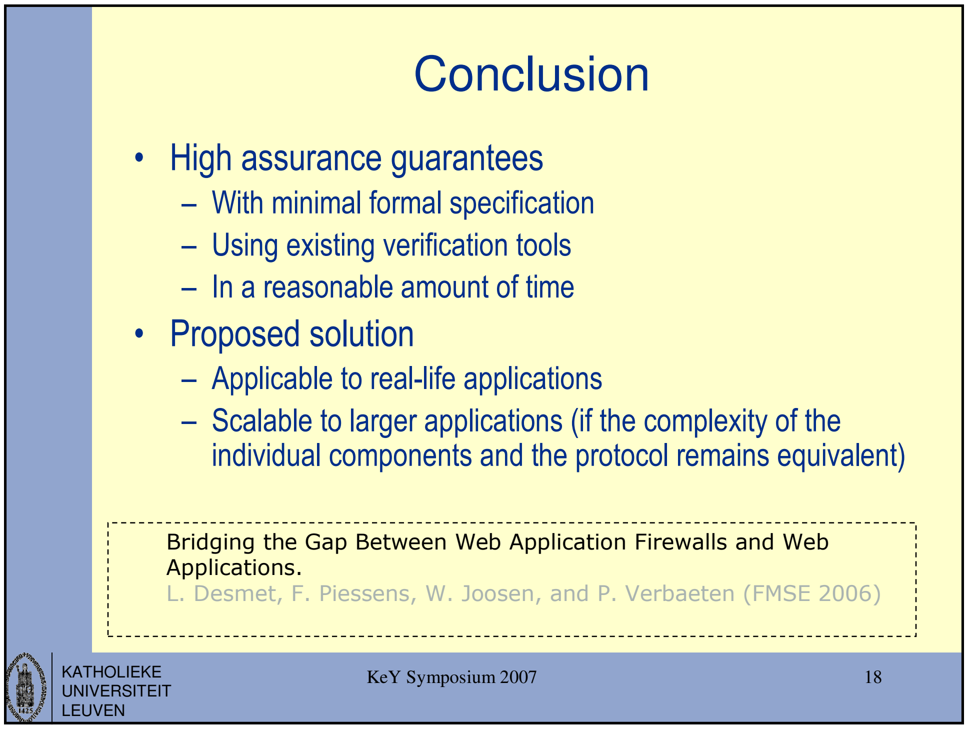
- Limit traffic to the intended client/server protocol
- Typical use of a Web Application Firewall (WAF) in protecting against forceful browsing



Experimental results

- Annotation overhead:
    - At most 4 lines per component
- Verification performance:
    - Static verification took at most 4 minutes per component
- Run-time overhead:
    - Experiment:
        - sequence of 1000 visitors
        - on average 6 requests per session
        - 2% of the users applied forceful browsing
    - Measured run-time overhead of 1.3%

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 17

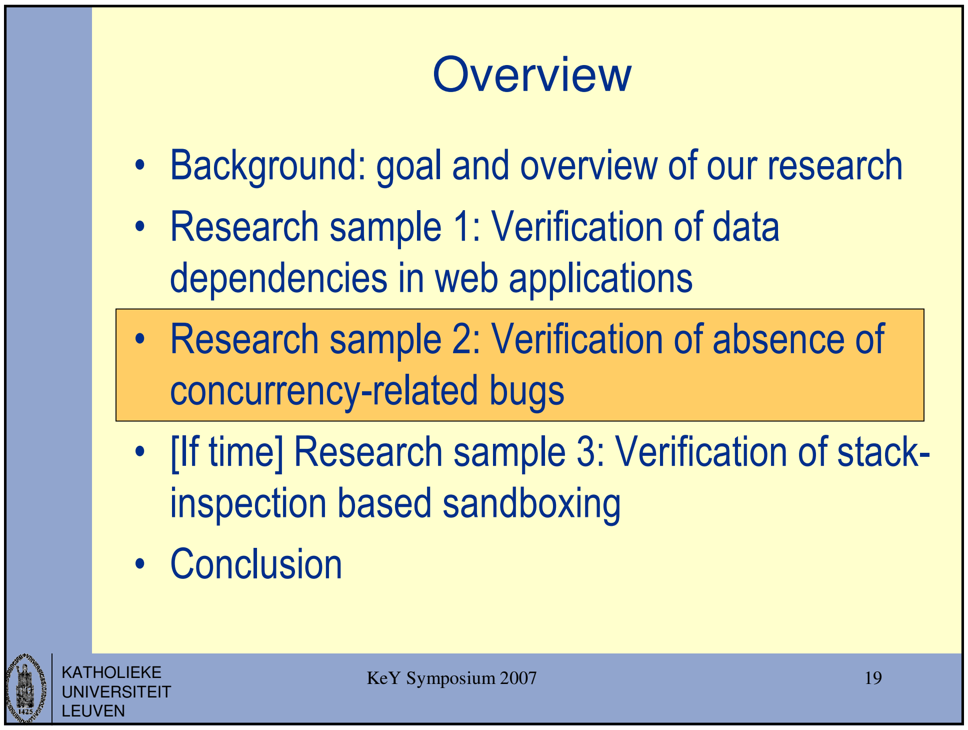


Conclusion

- High assurance guarantees
    - With minimal formal specification
    - Using existing verification tools
    - In a reasonable amount of time
- Proposed solution
    - Applicable to real-life applications
    - Scalable to larger applications (if the complexity of the individual components and the protocol remains equivalent)

Bridging the Gap Between Web Application Firewalls and Web Applications.
L. Desmet, F. Piessens, W. Joosen, and P. Verbaeten (FMSE 2006)

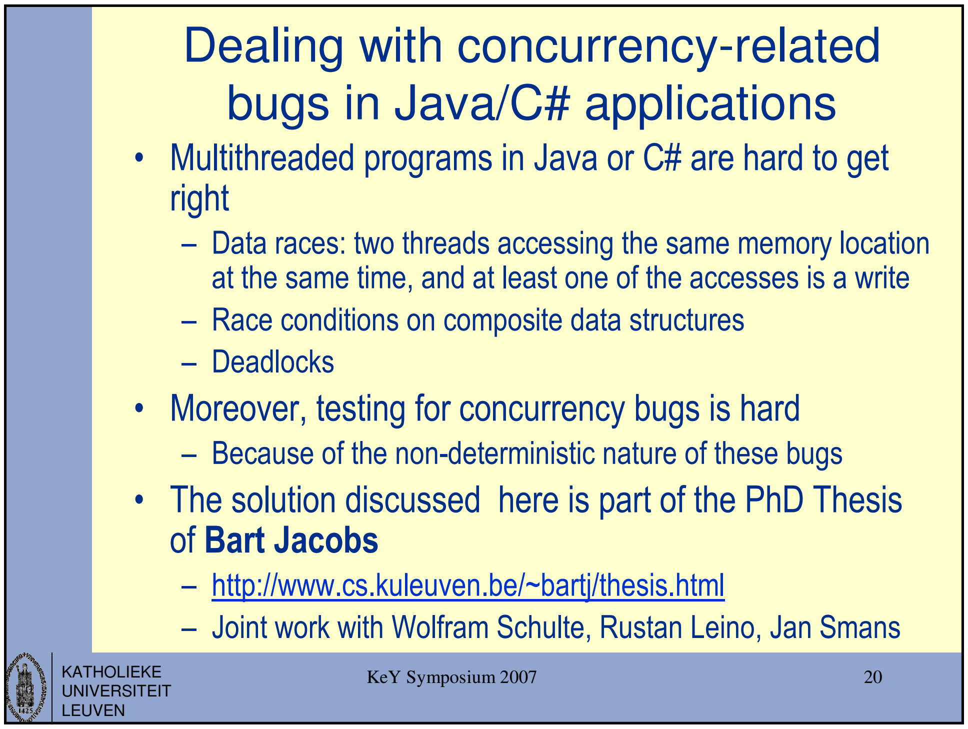
KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 18



Overview

- Background: goal and overview of our research
- Research sample 1: Verification of data dependencies in web applications
- Research sample 2: Verification of absence of concurrency-related bugs
- [If time] Research sample 3: Verification of stack-inspection based sandboxing
- Conclusion

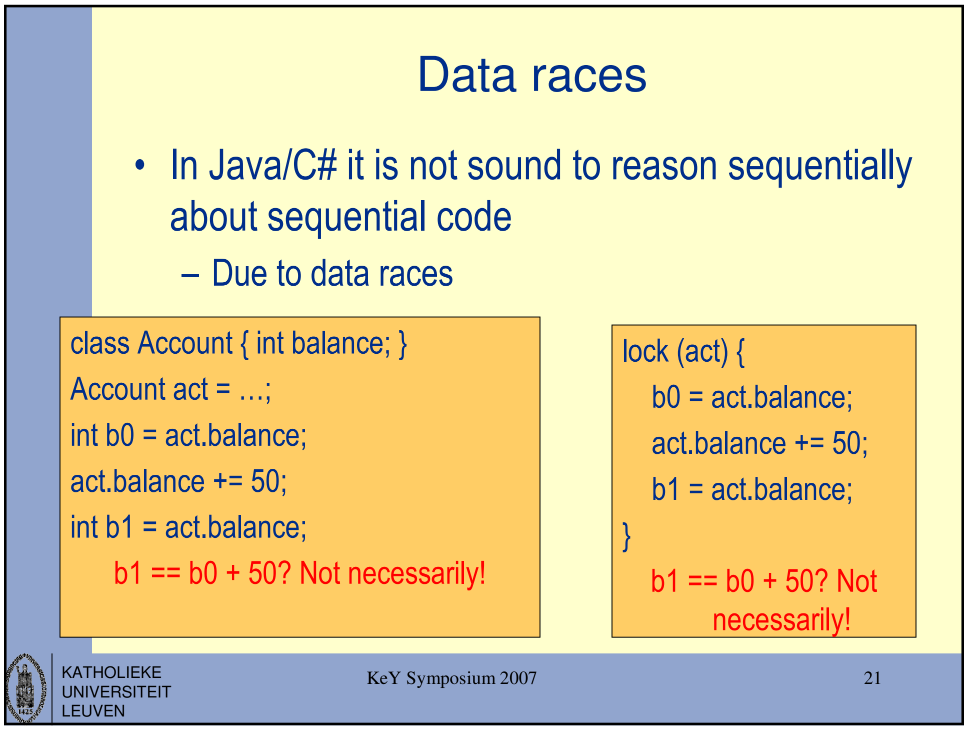
KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 19



Dealing with concurrency-related bugs in Java/C# applications

- Multithreaded programs in Java or C# are hard to get right
  - Data races: two threads accessing the same memory location at the same time, and at least one of the accesses is a write
  - Race conditions on composite data structures
  - Deadlocks
- Moreover, testing for concurrency bugs is hard
  - Because of the non-deterministic nature of these bugs
- The solution discussed here is part of the PhD Thesis of **Bart Jacobs**
  - http://www.cs.kuleuven.be/~bartj/thesis.html
  - Joint work with Wolfram Schulte, Rustan Leino, Jan Smans

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 20



Data races

- In Java/C# it is not sound to reason sequentially about sequential code
  - Due to data races

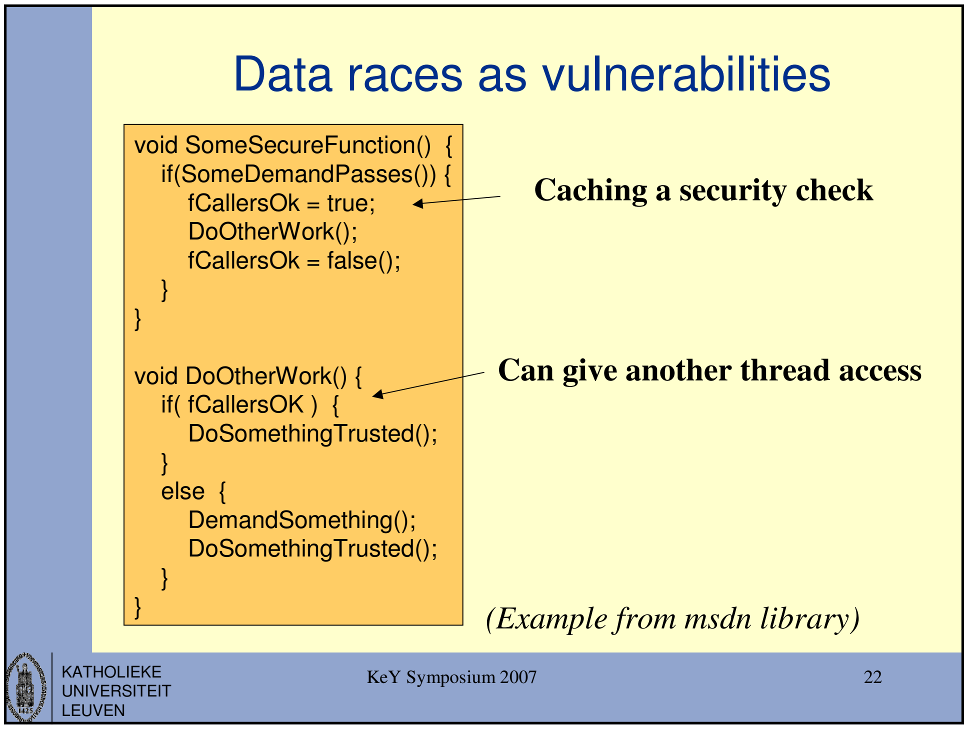
```
class Account { int balance; }
Account act = …;
int b0 = act.balance;
act.balance += 50;
int b1 = act.balance;
      b1 == b0 + 50? Not necessarily!
```

```
lock (act) {
   b0 = act.balance;
   act.balance += 50;
   b1 = act.balance;
}
   b1 == b0 + 50? Not
         necessarily!
```

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

21



Data races as vulnerabilities

```
void SomeSecureFunction()  {
   if(SomeDemandPasses()) {
      fCallersOk = true;
      DoOtherWork();
      fCallersOk = false();
   }
}

void DoOtherWork() {
   if( fCallersOK )  {
      DoSomethingTrusted();
   }
   else  {
      DemandSomething();
      DoSomethingTrusted();
   }
}
```

Caching a security check

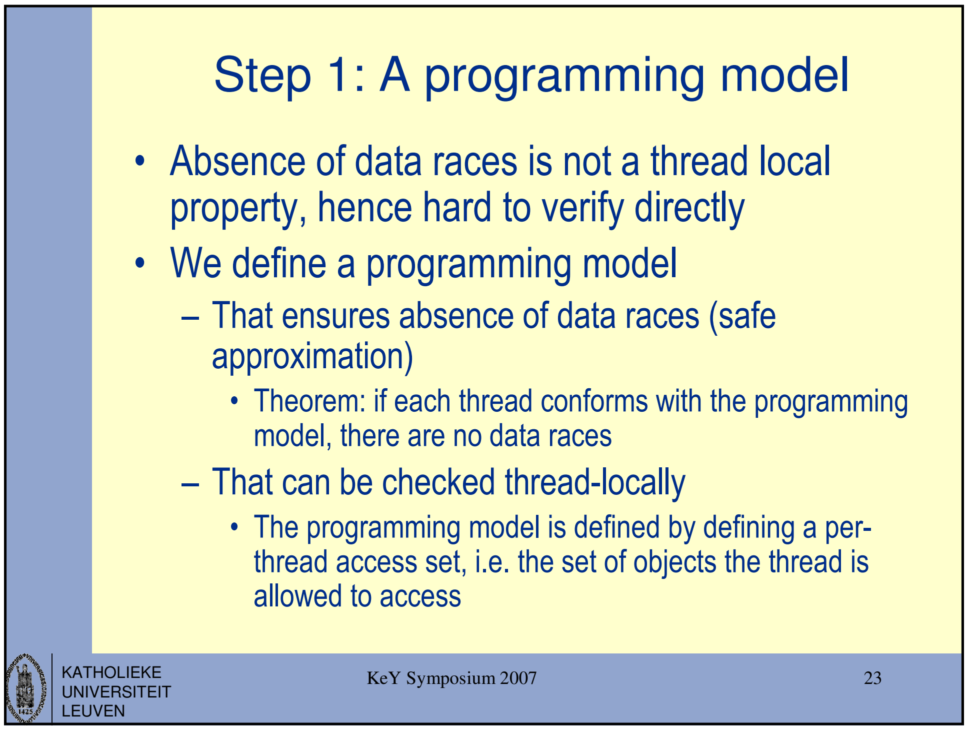
Can give another thread access

(Example from msdn library)

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

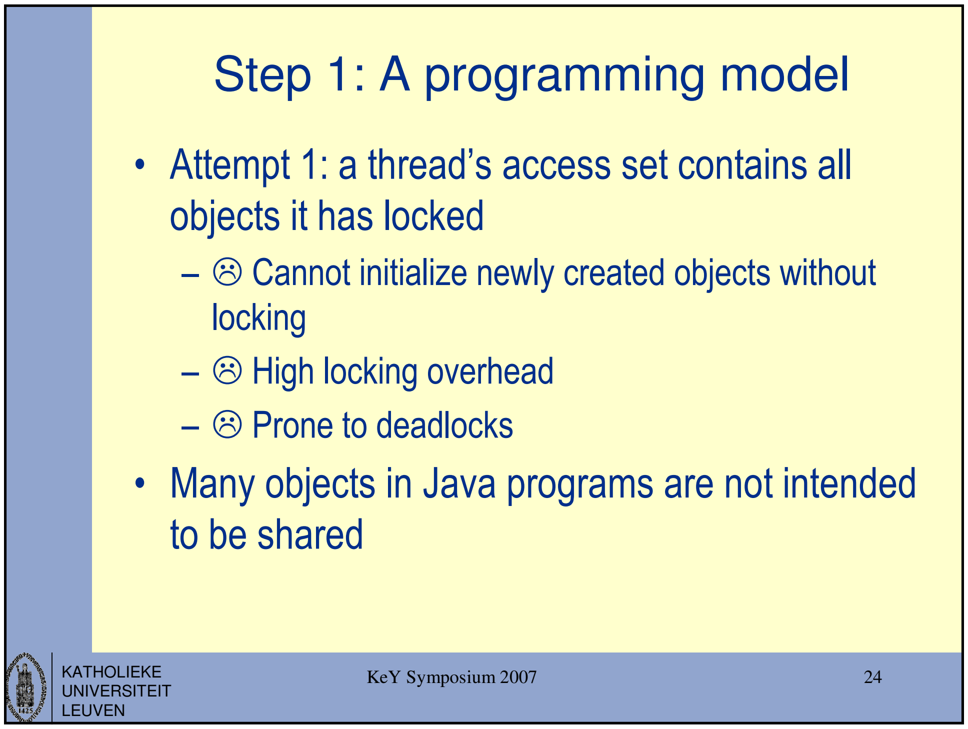
22



Step 1: A programming model

- Absence of data races is not a thread local property, hence hard to verify directly
- We define a programming model
  - That ensures absence of data races (safe approximation)
    - Theorem: if each thread conforms with the programming model, there are no data races
  - That can be checked thread-locally
    - The programming model is defined by defining a per-thread access set, i.e. the set of objects the thread is allowed to access

KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 23



Step 1: A programming model

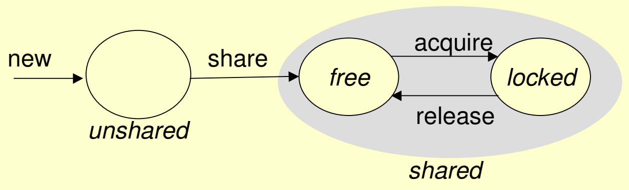
- Attempt 1: a thread's access set contains all objects it has locked
  - ☹ Cannot initialize newly created objects without locking
  - ☹ High locking overhead
  - ☹ Prone to deadlocks
- Many objects in Java programs are not intended to be shared

KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 24

Step 1: A programming model

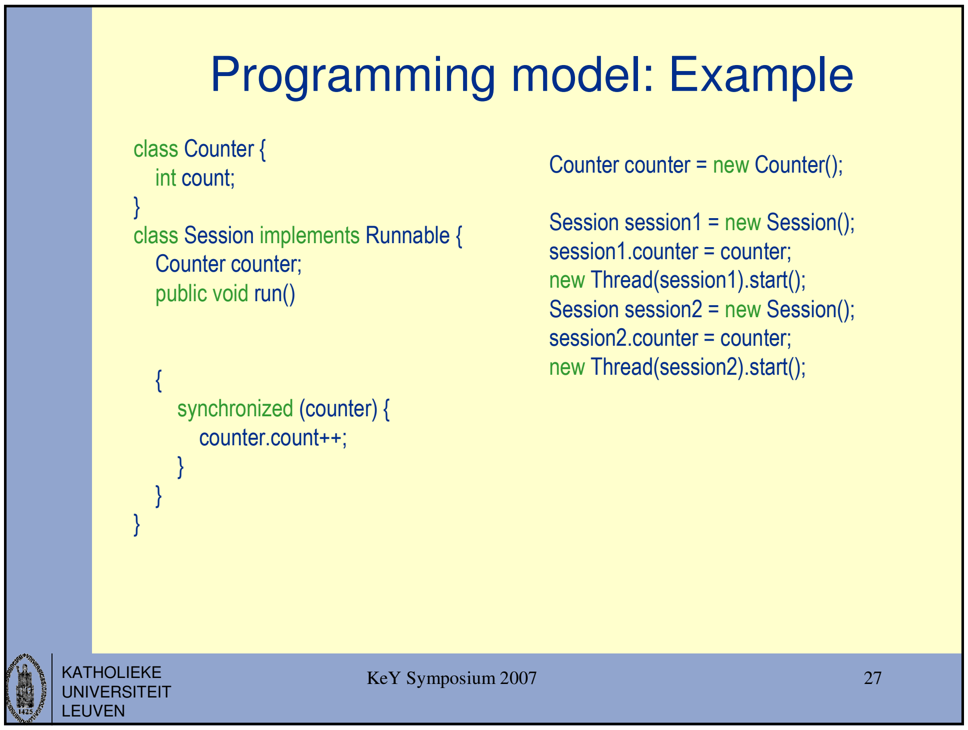
- Attempt 2: distinguish between **shared** and **thread-local** (unshared) objects
- Programmer has to explicitly indicate what objects are intended to be shared, with a **share o** operation

Object states



Rules of the programming model

1. Threads can only r/w fields of objects in their access set
2. New objects are unshared and element of the creating thread's access set
3. A **share o:** (only allowed if current thread has access to o, and o is unshared)
   - Removes o from the current thread's access set
   - Makes o shared
4. Entering a **synchronized (o):** (only allowed if o is shared)
   - Adds o to the current thread's access set
5. Leaving the synchronized block
   - Removes o again
6. Starting a new thread with runnable object o:
   - Transfers o to the access set of the new thread



Programming model: Example

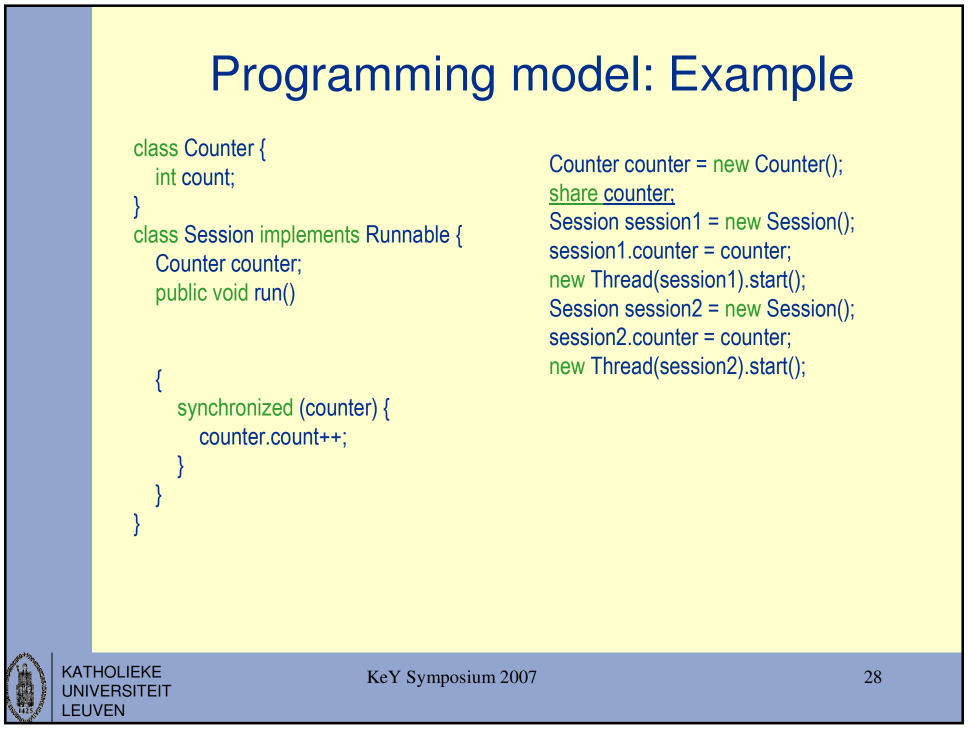
```
class Counter {
   int count;
}
class Session implements Runnable {
   Counter counter;
   public void run()


   {
      synchronized (counter) {
         counter.count++;
      }
   }
}
```

```
Counter counter = new Counter();

Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
```

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

27



Programming model: Example

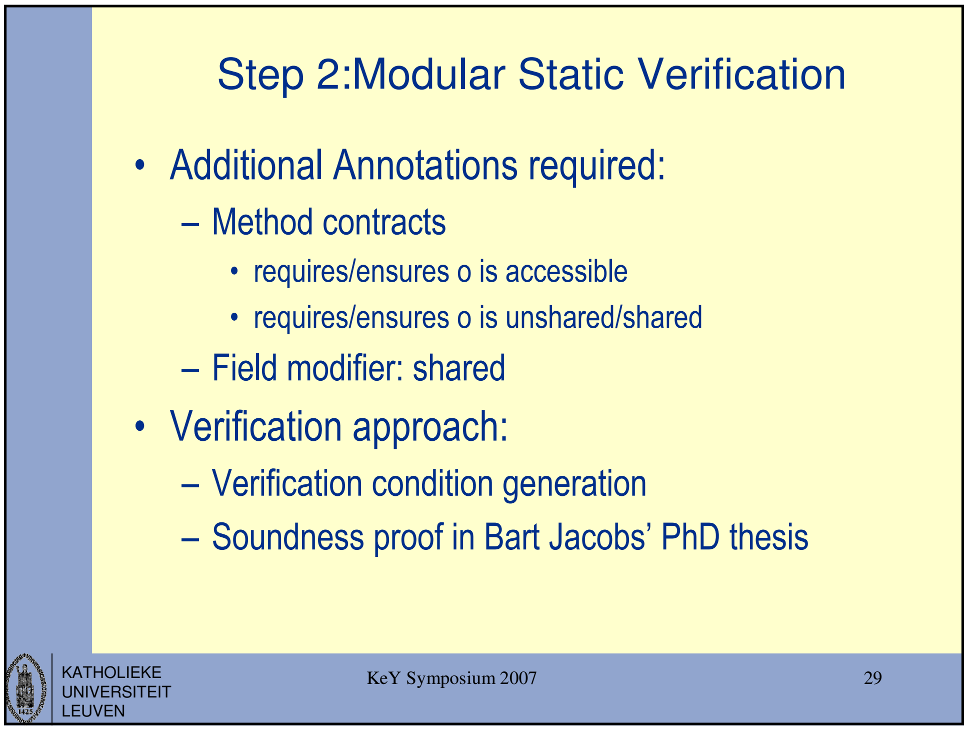
```
class Counter {
   int count;
}
class Session implements Runnable {
   Counter counter;
   public void run()


   {
      synchronized (counter) {
         counter.count++;
      }
   }
}
```

```
Counter counter = new Counter();
share counter;
Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
```

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

28



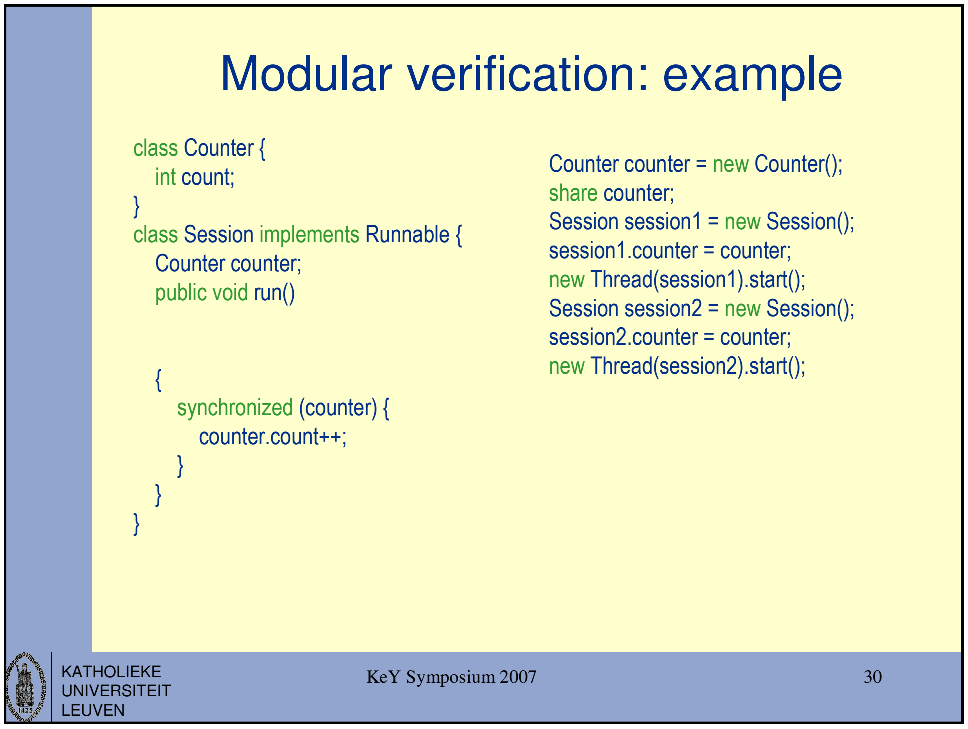
Step 2:Modular Static Verification

- Additional Annotations required:
  - Method contracts
    - requires/ensures o is accessible
    - requires/ensures o is unshared/shared
  - Field modifier: shared
- Verification approach:
  - Verification condition generation
  - Soundness proof in Bart Jacobs' PhD thesis

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

29



Modular verification: example

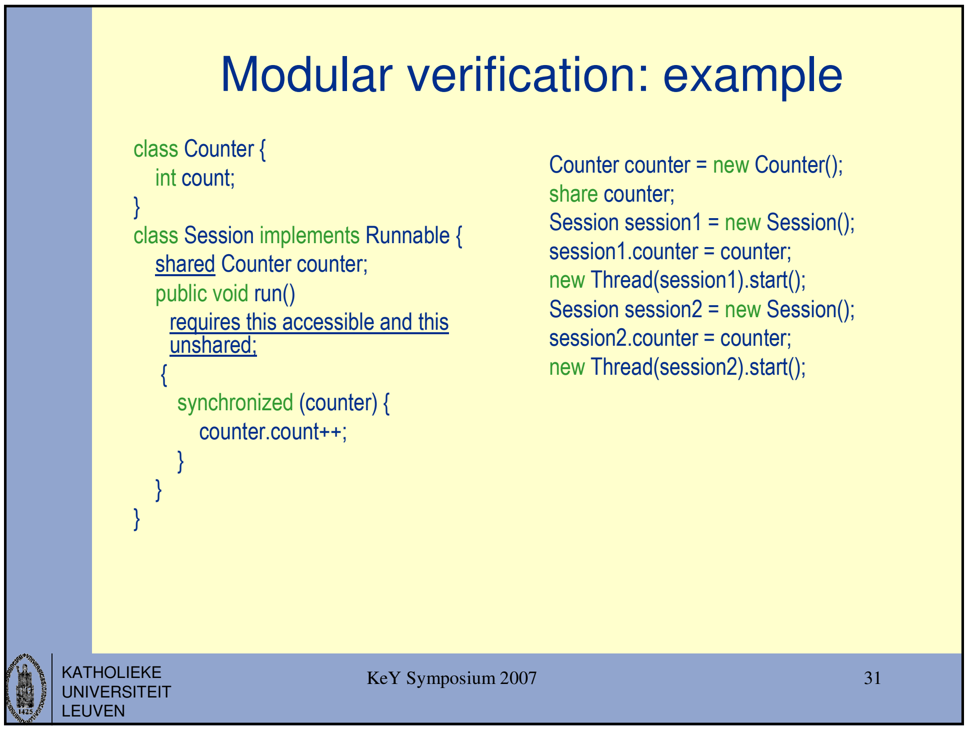
```
class Counter {
   int count;
}
class Session implements Runnable {
   Counter counter;
   public void run()


   {
      synchronized (counter) {
         counter.count++;
      }
   }
}
```

```
Counter counter = new Counter();
share counter;
Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
```

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007

30

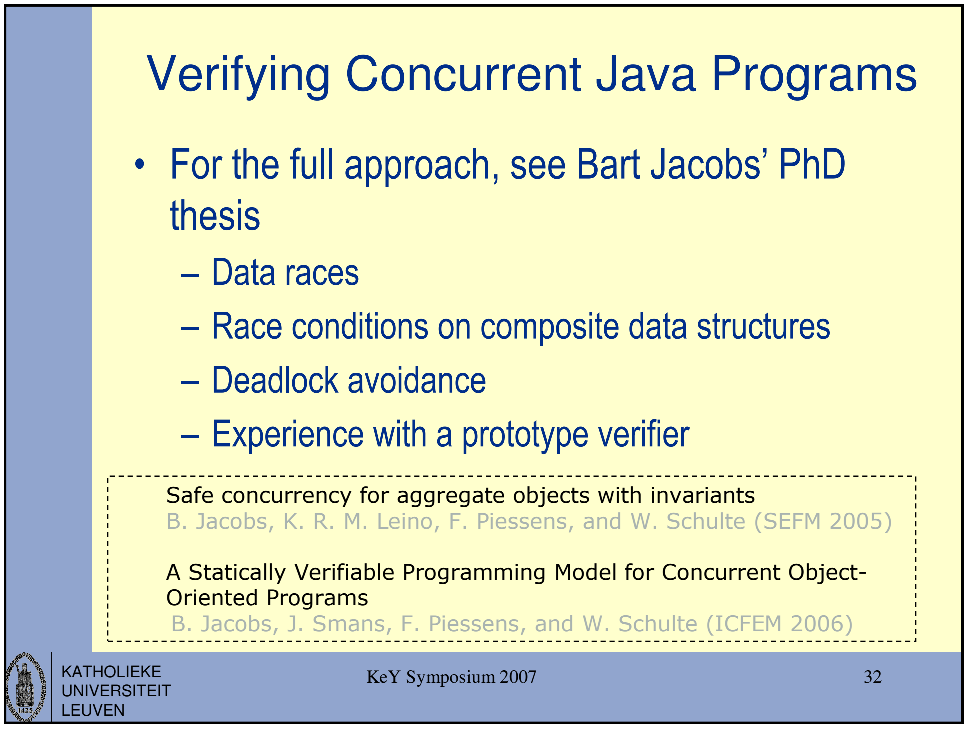


Modular verification: example

```
class Counter {
  int count;
}
class Session implements Runnable {
  shared Counter counter;
  public void run()
    requires this accessible and this
    unshared;
   {
     synchronized (counter) {
       counter.count++;
     }
  }
}
```

```
Counter counter = new Counter();
share counter;
Session session1 = new Session();
session1.counter = counter;
new Thread(session1).start();
Session session2 = new Session();
session2.counter = counter;
new Thread(session2).start();
```

KATHOLIEKE UNIVERSITEIT LEUVEN



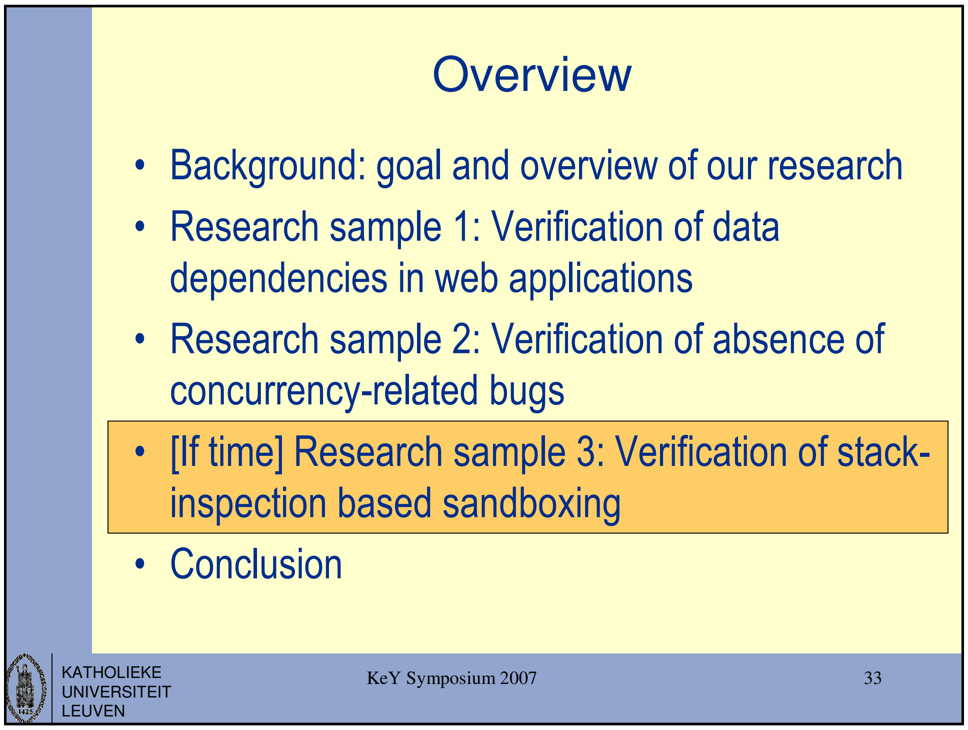
Verifying Concurrent Java Programs

- For the full approach, see Bart Jacobs' PhD thesis
  - Data races
  - Race conditions on composite data structures
  - Deadlock avoidance
  - Experience with a prototype verifier

Safe concurrency for aggregate objects with invariants
B. Jacobs, K. R. M. Leino, F. Piessens, and W. Schulte (SEFM 2005)

A Statically Verifiable Programming Model for Concurrent Object-Oriented Programs
B. Jacobs, J. Smans, F. Piessens, and W. Schulte (ICFEM 2006)

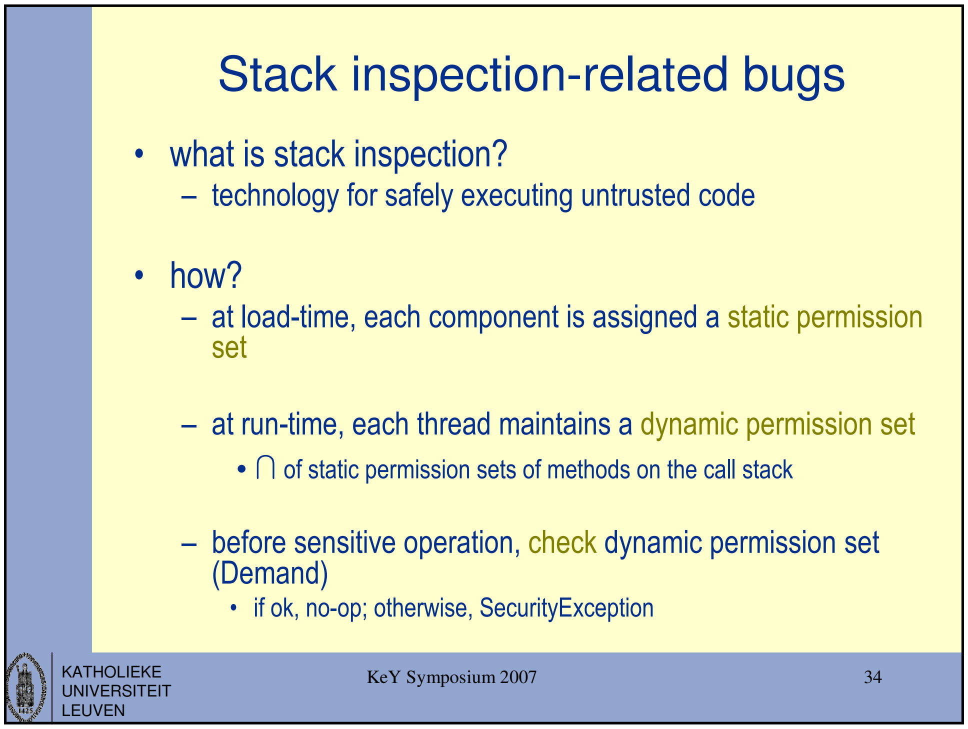
KATHOLIEKE UNIVERSITEIT LEUVEN



Overview

- Background: goal and overview of our research
- Research sample 1: Verification of data dependencies in web applications
- Research sample 2: Verification of absence of concurrency-related bugs
- [If time] Research sample 3: Verification of stack-inspection based sandboxing
- Conclusion

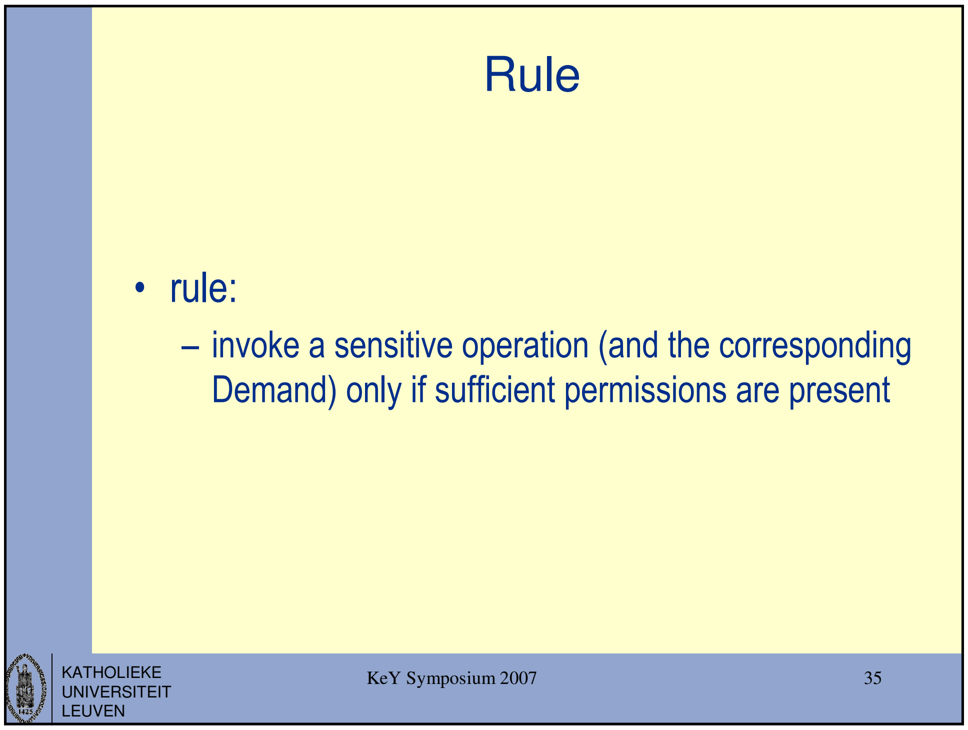
KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 33



Stack inspection-related bugs

- what is stack inspection?
  - technology for safely executing untrusted code
- how?
  - at load-time, each component is assigned a static permission set
  - at run-time, each thread maintains a dynamic permission set
    - $\bigcap$ of static permission sets of methods on the call stack
  - before sensitive operation, check dynamic permission set (Demand)
    - if ok, no-op; otherwise, SecurityException

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 34

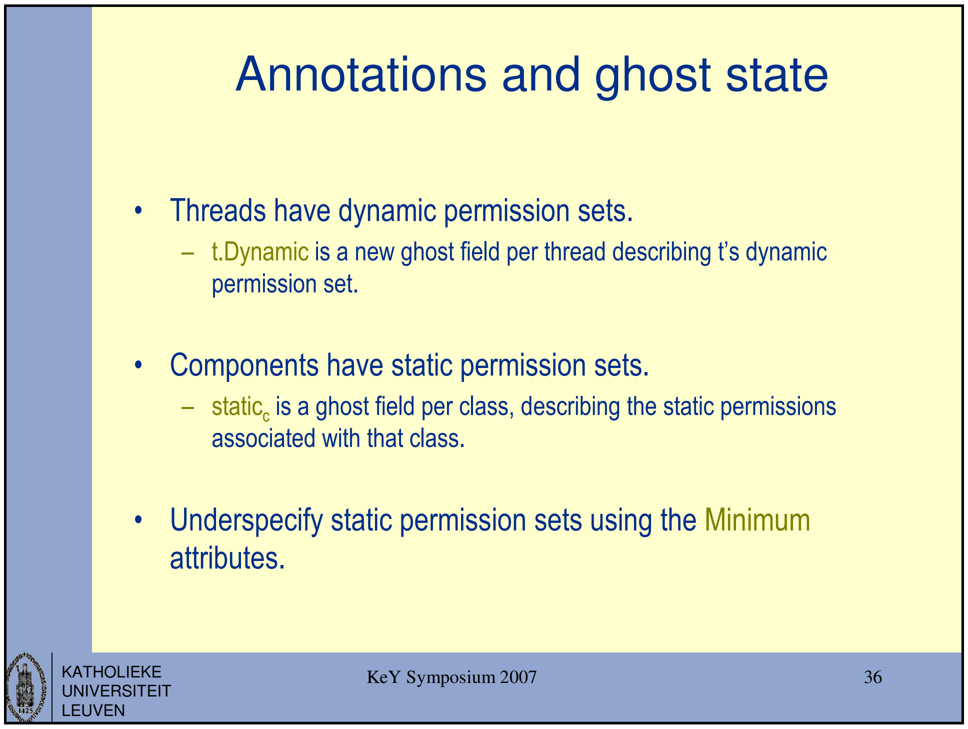


Rule

- rule:
  - invoke a sensitive operation (and the corresponding Demand) only if sufficient permissions are present

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007 35

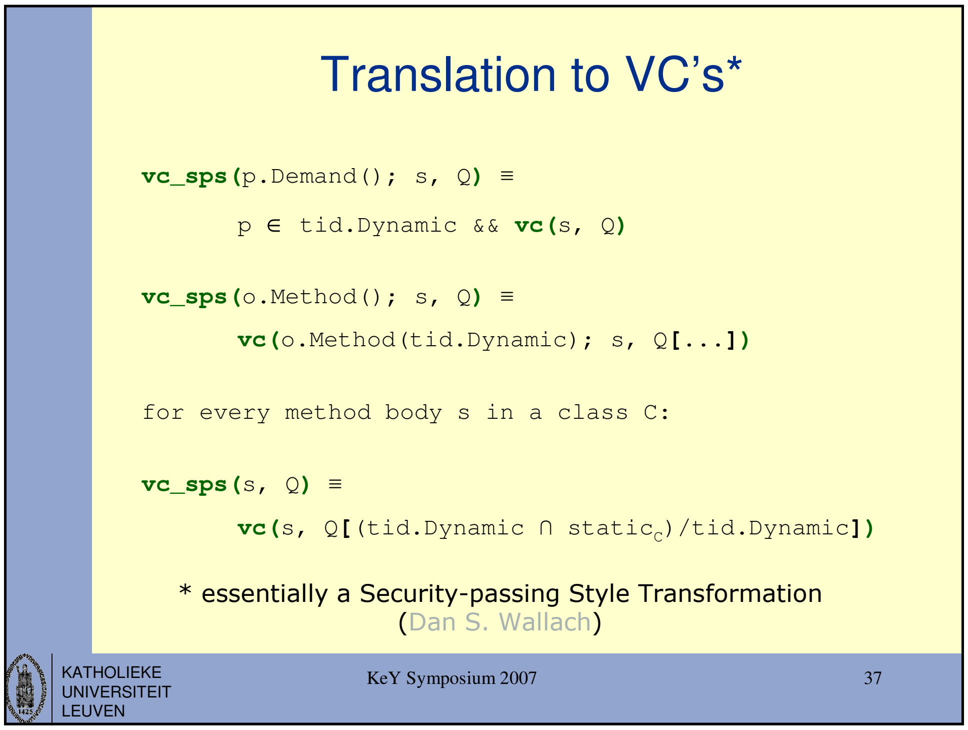


Annotations and ghost state

- Threads have dynamic permission sets.
  - t.Dynamic is a new ghost field per thread describing t's dynamic permission set.
- Components have static permission sets.
  - static_c is a ghost field per class, describing the static permissions associated with that class.
- Underspecify static permission sets using the Minimum attributes.

KATHOLIEKE UNIVERSITEIT LEUVEN

KeY Symposium 2007 36

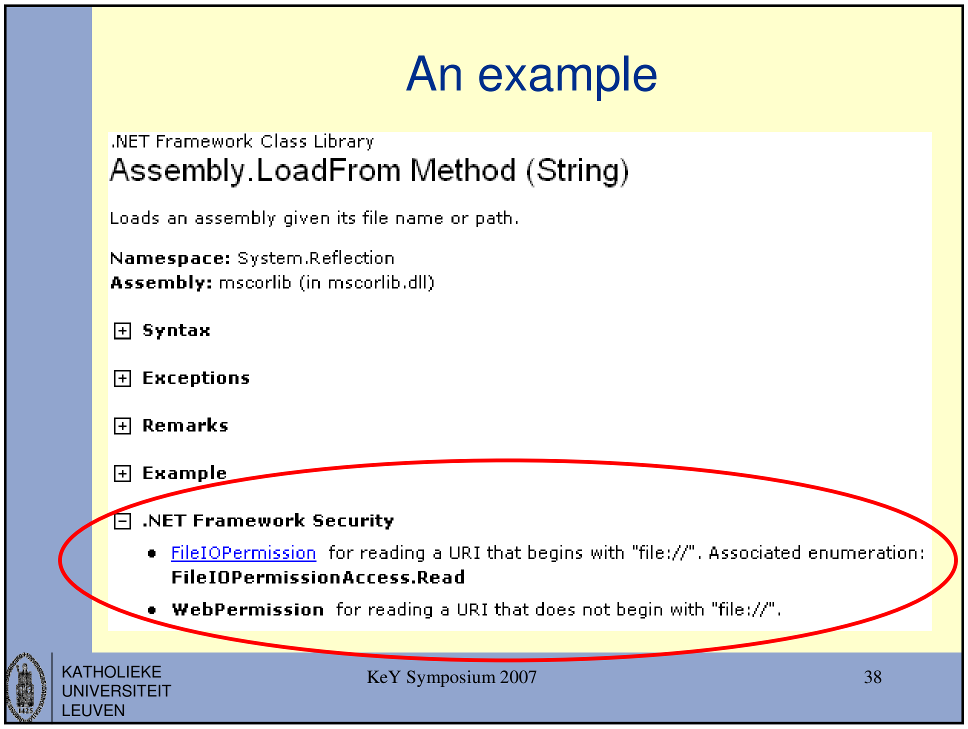


Translation to VC's*

```
vc_sps(p.Demand(); s, Q) ≡
      p ∈ tid.Dynamic && vc(s, Q)

vc_sps(o.Method(); s, Q) ≡
      vc(o.Method(tid.Dynamic); s, Q[...])

for every method body s in a class C:

vc_sps(s, Q) ≡
      vc(s, Q[(tid.Dynamic ∩ static_C)/tid.Dynamic])
```

* essentially a Security-passing Style Transformation
(Dan S. Wallach)

KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 37



An example

.NET Framework Class Library

Assembly.LoadFrom Method (String)

Loads an assembly given its file name or path.

Namespace: System.Reflection
Assembly: mscorlib (in mscorlib.dll)

⊞ **Syntax**

⊞ **Exceptions**

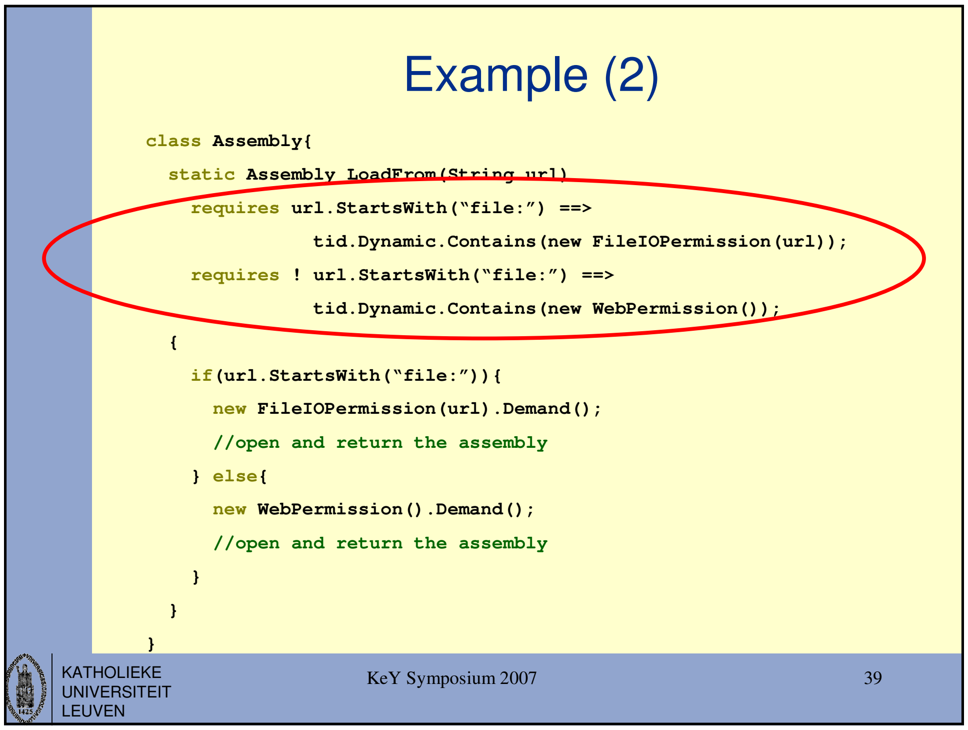
⊞ **Remarks**

⊞ **Example**

⊟ **.NET Framework Security**

- FileIOPermission for reading a URI that begins with "file://". Associated enumeration: **FileIOPermissionAccess.Read**
- **WebPermission** for reading a URI that does not begin with "file://".

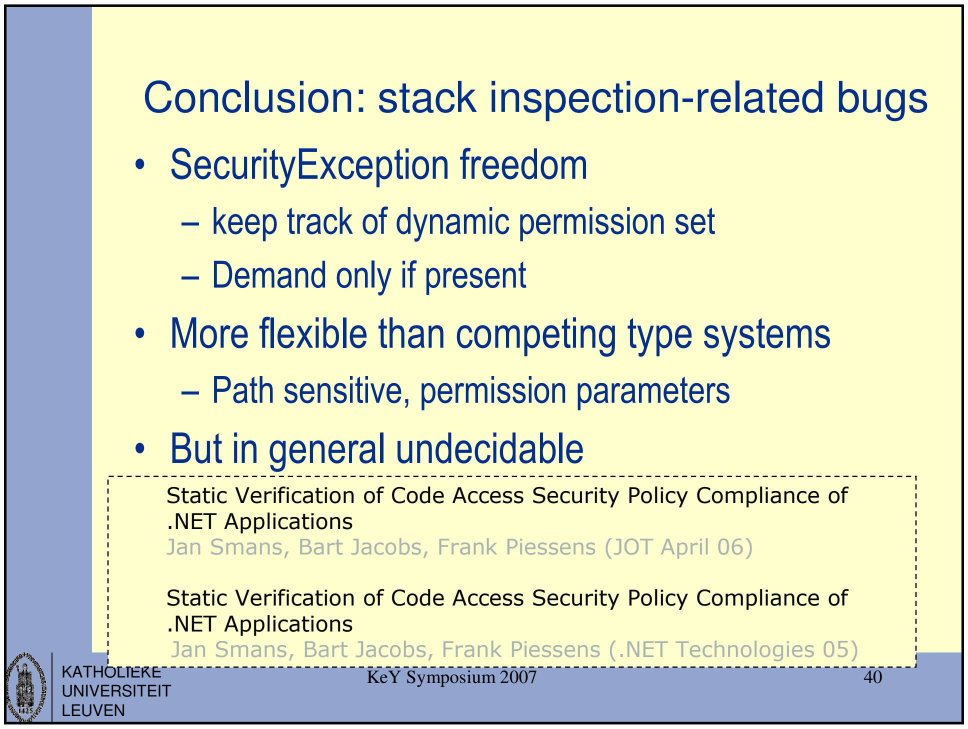
KATHOLIEKE UNIVERSITEIT LEUVEN — KeY Symposium 2007 — 38



Example (2)

```
class Assembly{
  static Assembly LoadFrom(String url)
    requires url.StartsWith("file:") ==>
                tid.Dynamic.Contains(new FileIOPermission(url));
    requires ! url.StartsWith("file:") ==>
                tid.Dynamic.Contains(new WebPermission());
  {
    if(url.StartsWith("file:")){
      new FileIOPermission(url).Demand();
      //open and return the assembly
    } else{
      new WebPermission().Demand();
      //open and return the assembly
    }
  }
}
```

KATHOLIEKE UNIVERSITEIT LEUVEN   KeY Symposium 2007   39



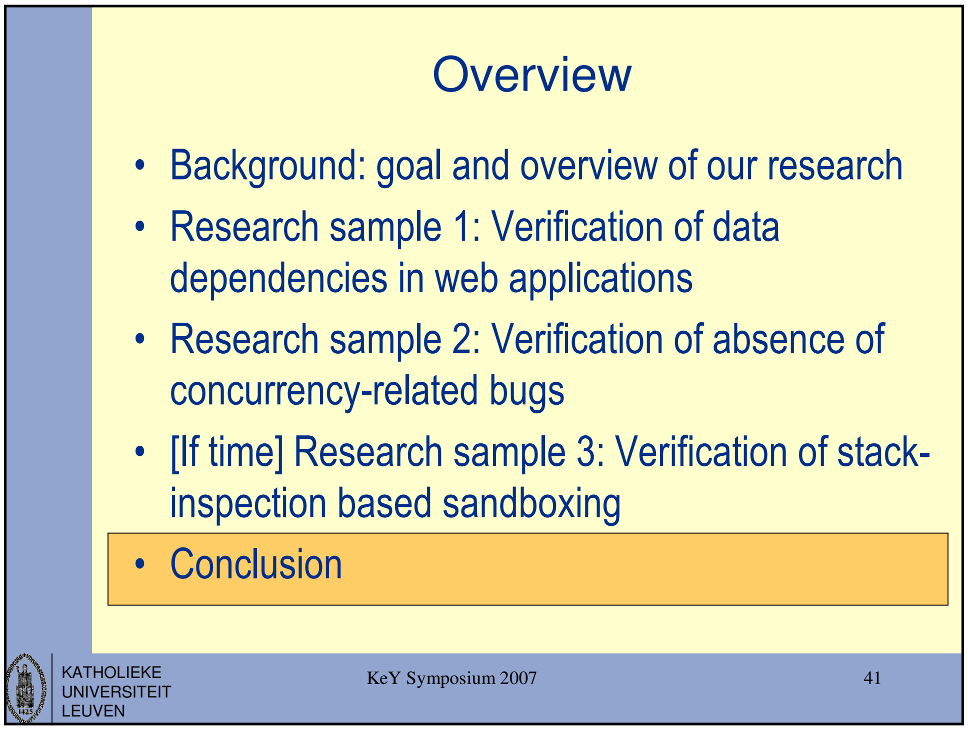
Conclusion: stack inspection-related bugs

- SecurityException freedom
  - keep track of dynamic permission set
  - Demand only if present
- More flexible than competing type systems
  - Path sensitive, permission parameters
- But in general undecidable

Static Verification of Code Access Security Policy Compliance of .NET Applications
Jan Smans, Bart Jacobs, Frank Piessens (JOT April 06)

Static Verification of Code Access Security Policy Compliance of .NET Applications
Jan Smans, Bart Jacobs, Frank Piessens (.NET Technologies 05)

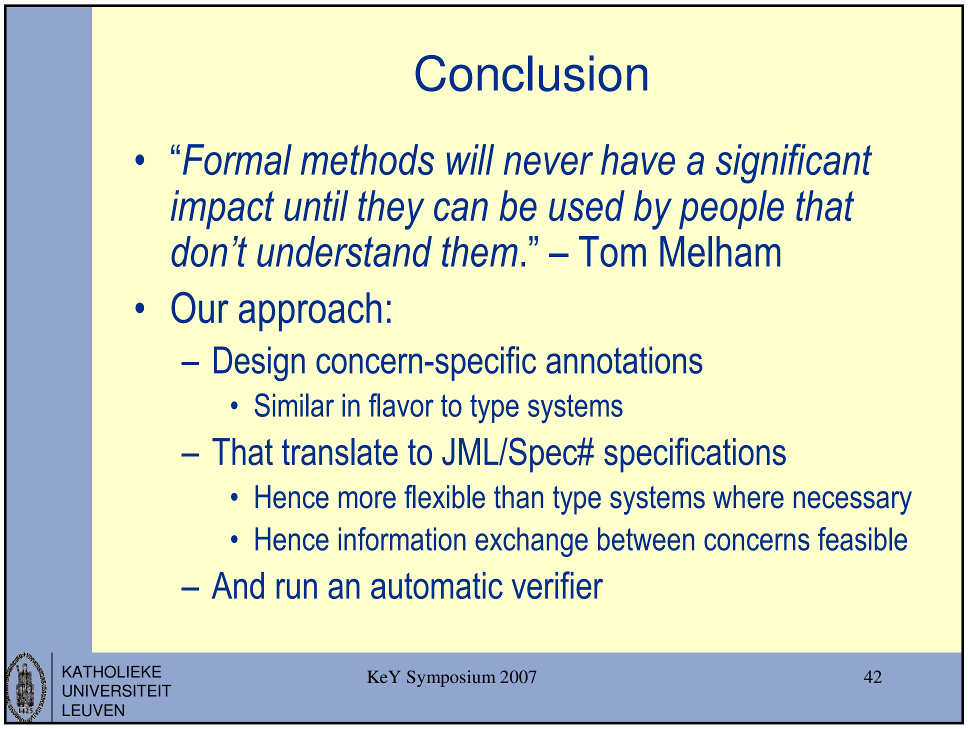
KATHOLIEKE UNIVERSITEIT LEUVEN   KeY Symposium 2007   40



Overview

- Background: goal and overview of our research
- Research sample 1: Verification of data dependencies in web applications
- Research sample 2: Verification of absence of concurrency-related bugs
- [If time] Research sample 3: Verification of stack-inspection based sandboxing
- Conclusion

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 41



Conclusion

- "*Formal methods will never have a significant impact until they can be used by people that don't understand them.*" – Tom Melham
- Our approach:
  - Design concern-specific annotations
    - Similar in flavor to type systems
  - That translate to JML/Spec# specifications
    - Hence more flexible than type systems where necessary
    - Hence information exchange between concerns feasible
  - And run an automatic verifier

KATHOLIEKE UNIVERSITEIT LEUVEN KeY Symposium 2007 42