LAMINAR: PRACTICAL FINE-GRAINED DECENTRALIZED INFORMATION FLOW CONTROL (DIFC)

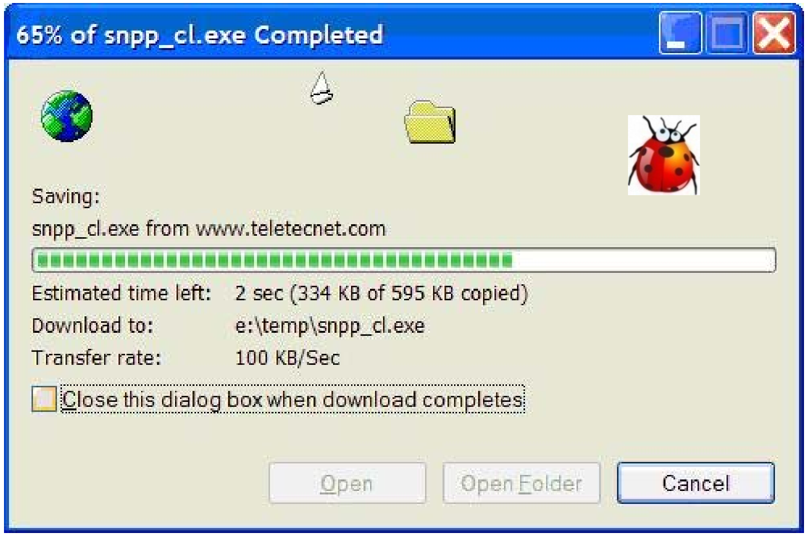
Indrajit Roy, Donald E. Porter, Michael D. Bond,

Kathryn S. McKinley, Emmett Witchel

The University of Texas at Austin

Untrusted code on trusted data

- Your computer holds trusted and sensitive data
  - Credit card number, SSN, personal calendar…
- But not every program you run is trusted
  - Bugs in code, malicious plugins…


65% of snpp_cl.exe Completed

Saving:
snpp_cl.exe from www.teletecnet.com

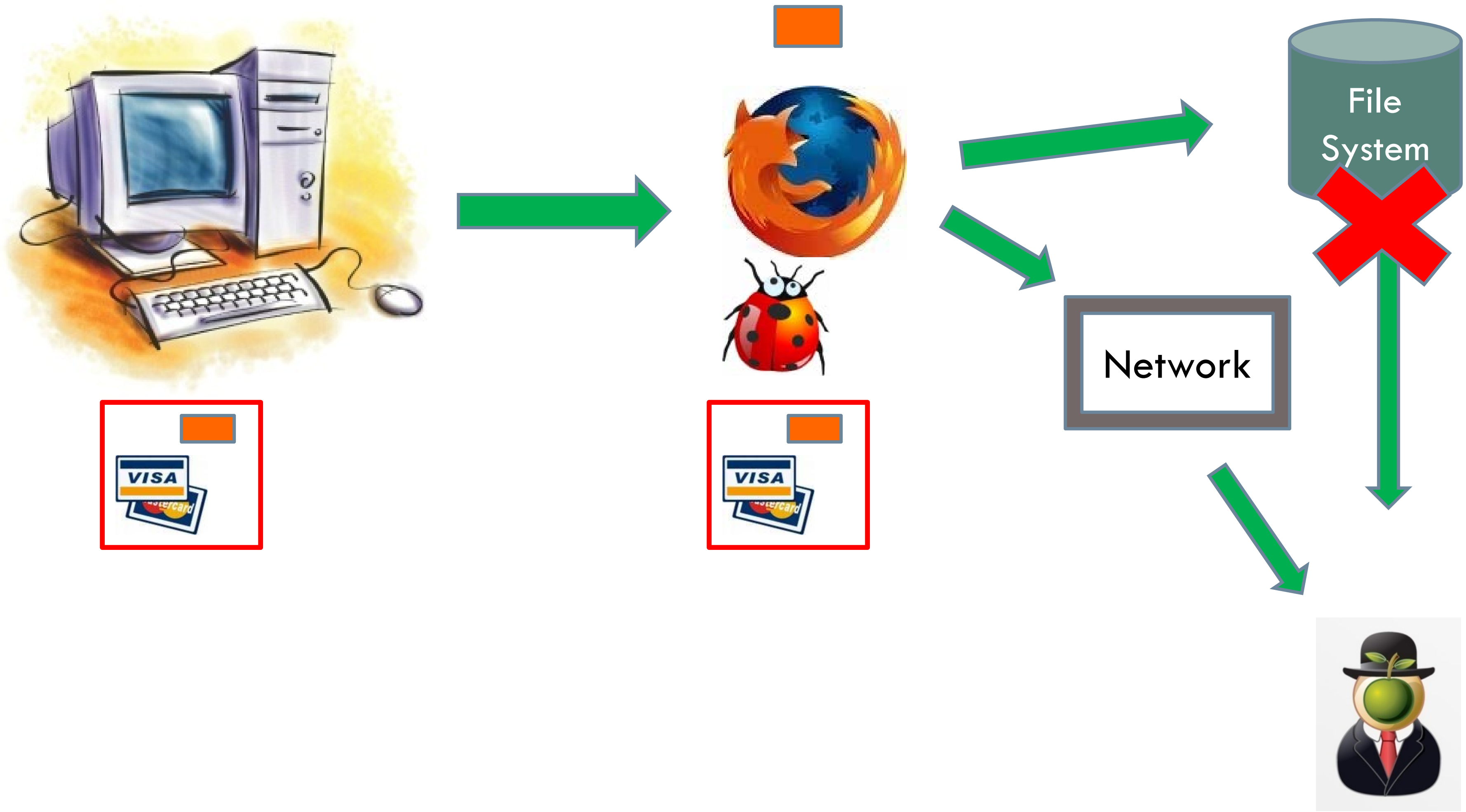
Estimated time left: 2 sec (334 KB of 595 KB copied)
Download to: e:\temp\snpp_cl.exe
Transfer rate: 100 KB/Sec

□ Close this dialog box when download completes

Security model

- Decentralized Information Flow Control (DIFC) [Myers and Liskov '97]
- Associate labels with the data
- System tracks the flow of data and the labels
- Access and distribution of data depends on labels
  - Firefox may read the credit card number
  - But firefox may **not** send it to the outside world

Control thy data (and its fate)

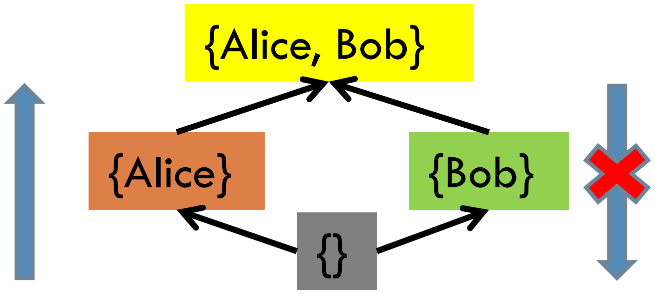


File System

Network

DIFC Implementation

- How do we rethink and rewrite code for security?
  - Hopefully not many changes…
- Users create a lattice of labels
- Associate labels with the data-structure



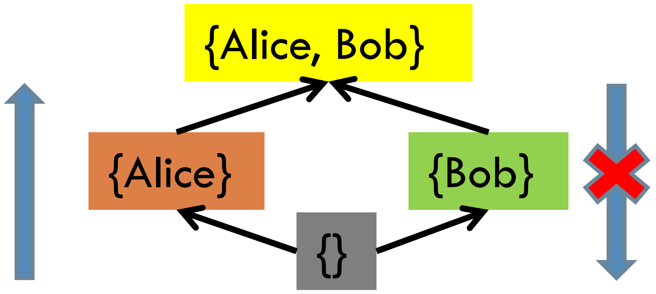
Information flow in a lattice

User	Mon.	Tue.	Wed.
Alice	Watch game	Office work	Free
Bob	Free	Meet doctor	Free

Calendar data-structure

Challenge: Programmability vs. security

- An ideal DIFC system
  - No code refactoring or changes to the data structures
  - Naturally interact with the file system and the network
  - Enforce fine-grained policies



Information flow in a lattice

User	Mon.	Tue.	Wed.
Alice	Watch game	Office work	Free
Bob	Free	Meet doctor	Free

Calendar data-structure

In this talk: Laminar

A practical way to provide end-to-end security guarantees.

Outline

- Comparison with current DIFC systems
- Laminar: programming model
  - Design: PL + OS techniques
  - Security regions
- Case studies and evaluation
- Summary

Current DIFC enabled systems

Two broad categories

- Programming language based (PL)
  - Example: Jif, Flow Caml
- Operating system based (OS)
  - Example: Asbestos, HiStar, Flume

Advantages of Laminar

	PL Based	OS based	Laminar
Fine grained	✓	✗	✓

Object level

Address space or page level

Advantages of Laminar

	PL Based	OS based	Laminar
Fine grained	✓	✗	✓
End-to-end guarantee	✗	✓	✓

Information leaks possible through files and sockets

Advantages of Laminar

	PL Based	OS based	Laminar
Fine grained	✓	✗	✓
End-to-end guarantee	✗	✓	✓
Incrementally deployable	✗	✗	✓

New language or type system

Code refactoring

Advantages of Laminar

	PL Based	OS based	Laminar
Fine grained	✓	✗	✓
End-to-end guarantee	✗	✓	✓
Incrementally deployable	✗	✗	✓
Advanced language features *	✗	✓	✓

*Dynamic class loading, reflection, multi-threading

Advantages of Laminar

	PL Based	OS based	Laminar
Fine grained	✓	✗	✓
End-to-end guarantee	✗	✓	✓
Incrementally deployable	✗	✗	✓
Advanced language features	✗	✓	✓

JVM tracks labels of objects

Dynamic analysis

JVM+OS integration

Security regions (new PL construct)

Outline

- Comparison with current DIFC systems
- Laminar: programming model
  - Design: PL + OS techniques
  - Security regions
- Case studies and evaluation
- Summary

Programming model

- No modifications to code that does not access the calendar
  - No need to trust such code!

User	Monday	Tuesday
Alice	Watch game	Office work
Bob	Free	Meet doctor

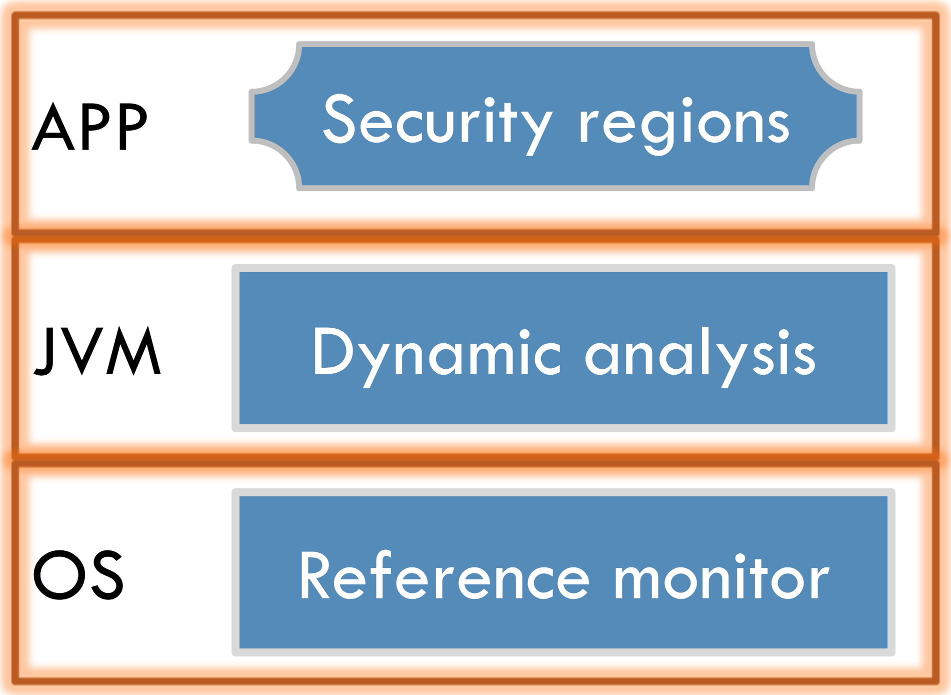
- Security regions
  - Wraps the code that accesses the calendar
  - Again, no need to trust the code!
    - Unless it modifies the labels of the data structure

Less work by the programmer.
Laminar enforces user security policy.

Trust assumptions

- Laminar JVM and Laminar OS should perform the correct DIFC checks
- Programmers should correctly specify the security policies using labels
- Limitation — covert channels
  - Timing channels
  - Termination channels
  - Probabilistic channels

Laminar design

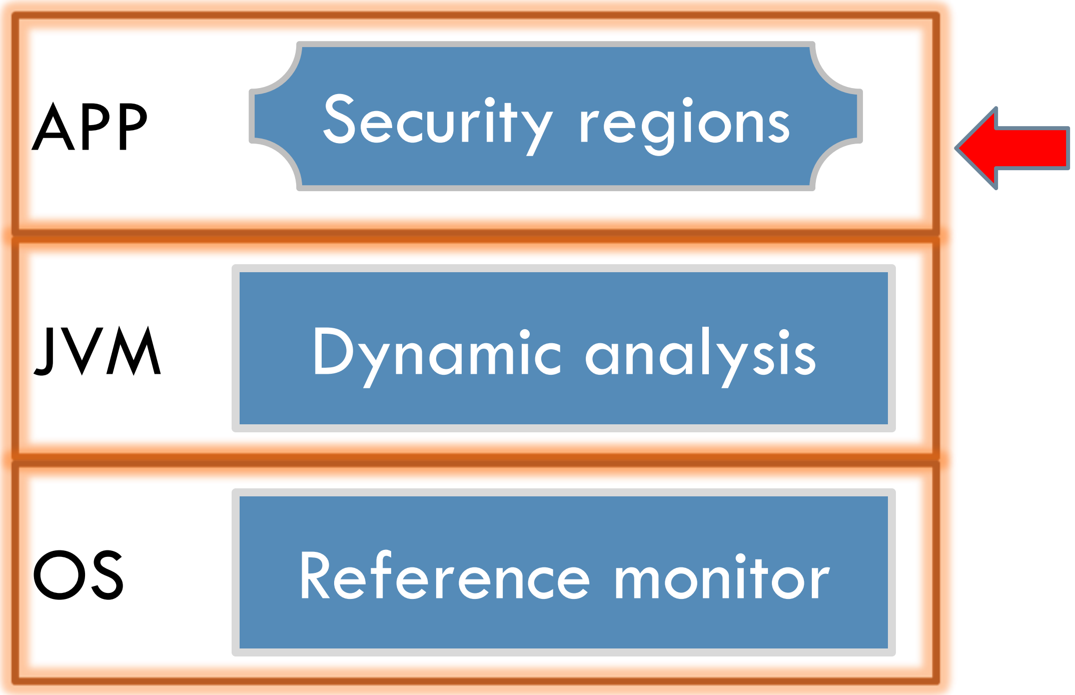


APP Security regions

JVM Dynamic analysis

OS Reference monitor

Laminar design: security regions



APP

Security regions

JVM

Dynamic analysis

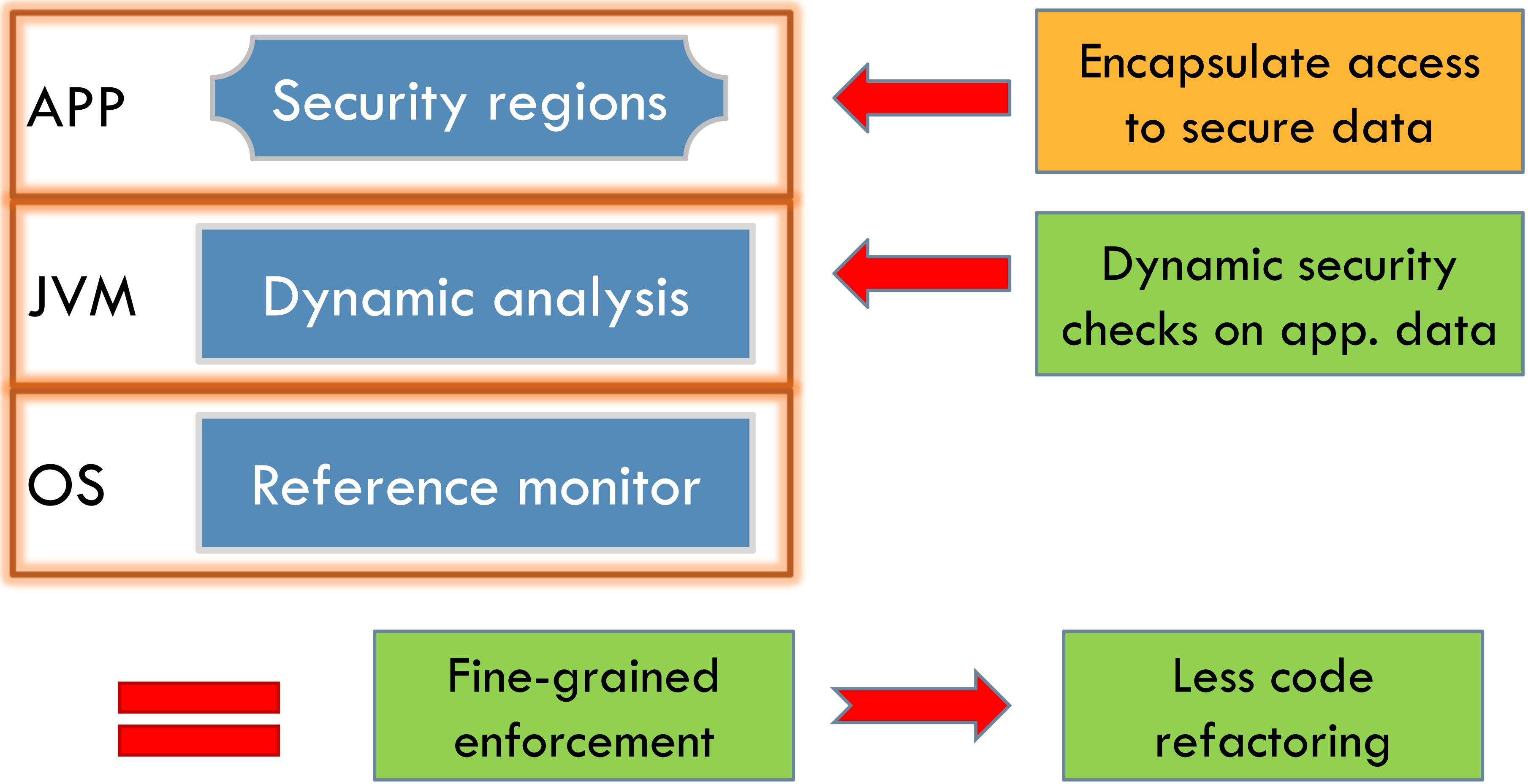
OS

Reference monitor

- Programming language construct
- Security sensitive data accessed only inside a security region

= Lowers overhead of DIFC checks + Helps incremental deployment

Laminar design: JVM



APP

Security regions

JVM

Dynamic analysis

OS

Reference monitor

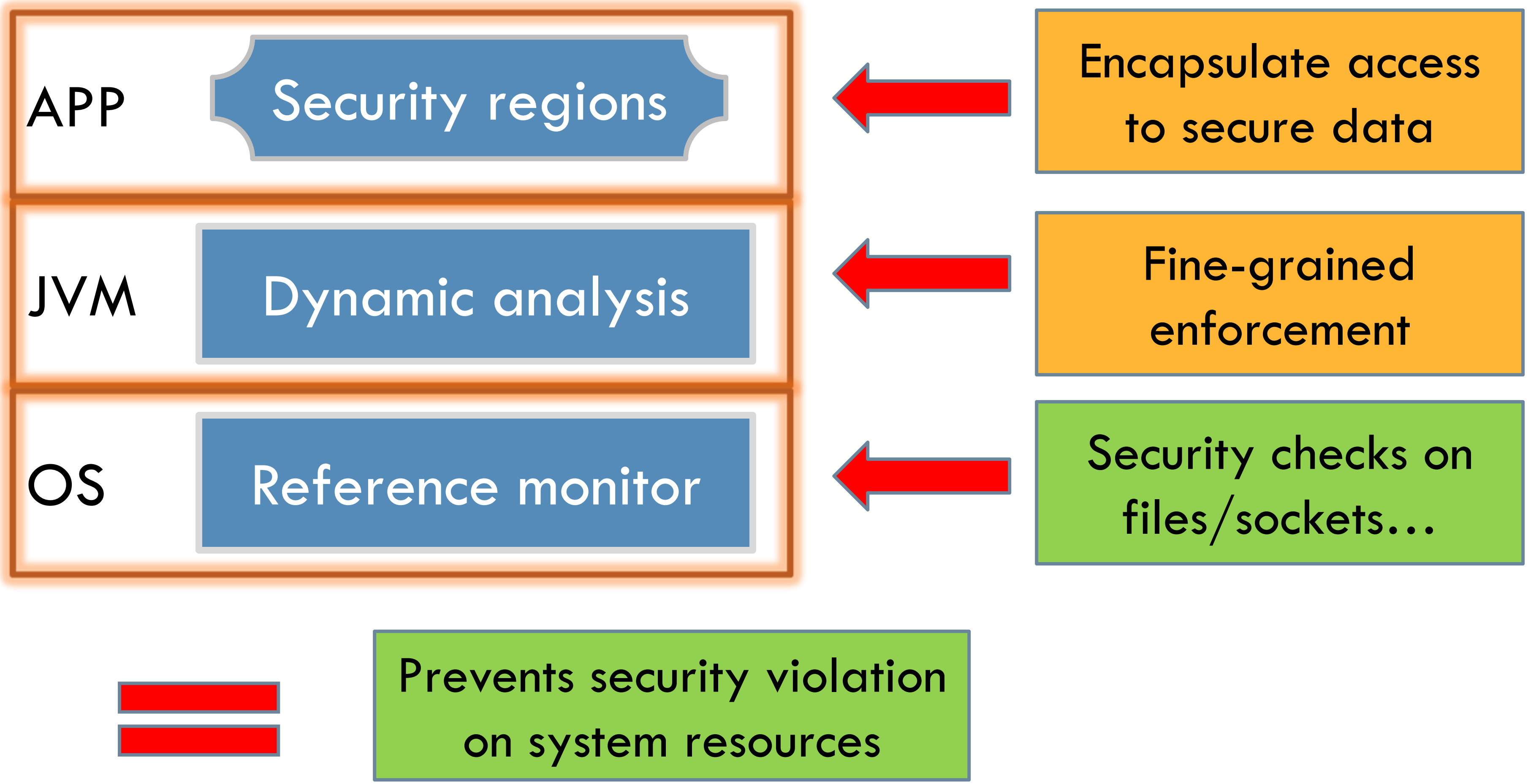
Encapsulate access to secure data

Dynamic security checks on app. data

Fine-grained enforcement

Less code refactoring

Laminar design : OS



APP

Security regions

Encapsulate access to secure data

JVM

Dynamic analysis

Fine-grained enforcement

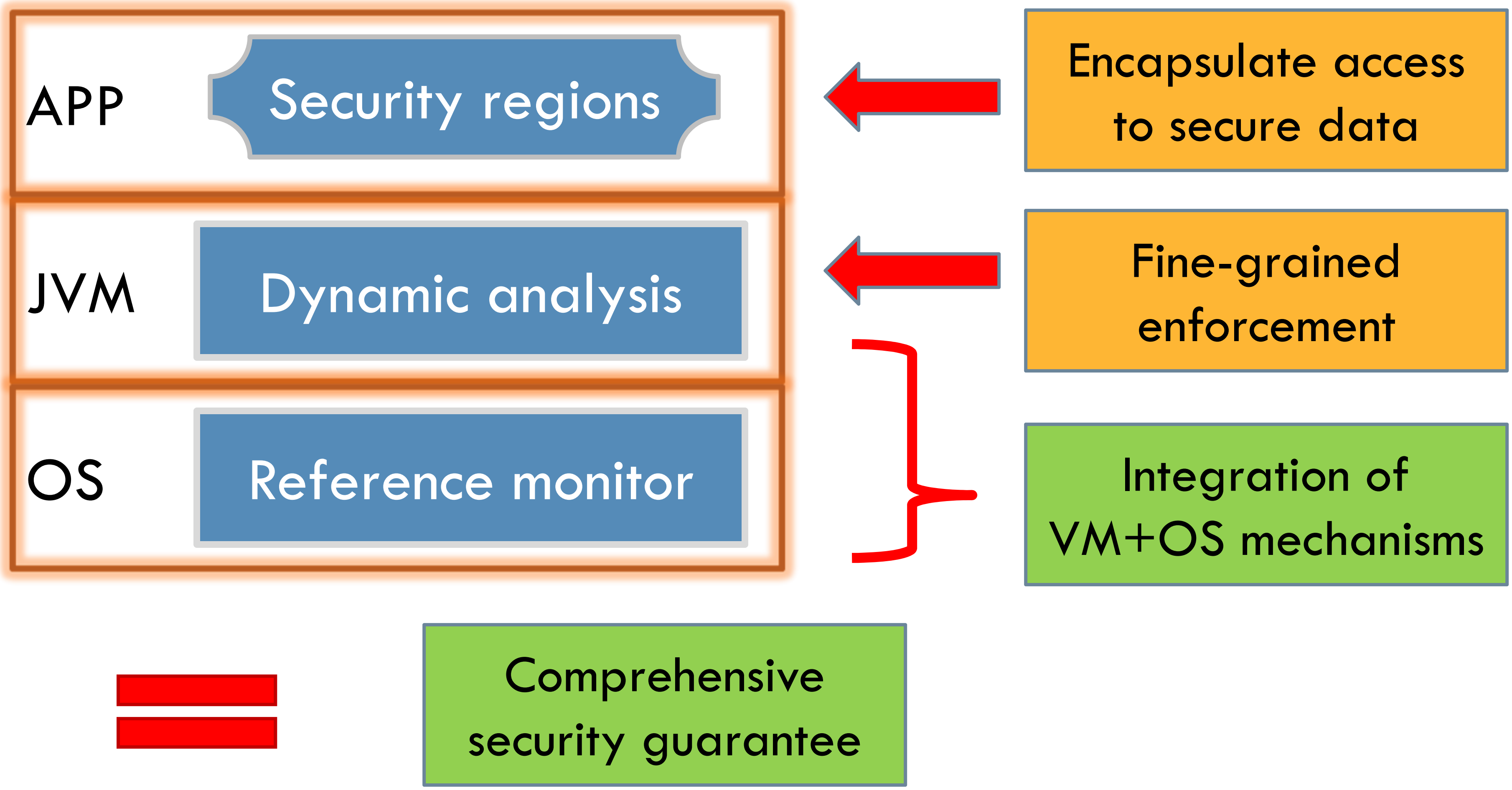
OS

Reference monitor

Security checks on files/sockets…

Prevents security violation on system resources

Laminar design : JVM+OS

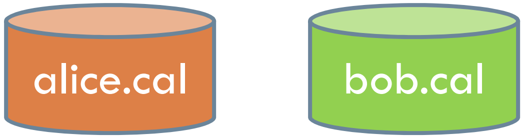

APP
Security regions
JVM
Dynamic analysis
OS
Reference monitor
Encapsulate access to secure data
Fine-grained enforcement
Integration of VM+OS mechanisms
=
Comprehensive security guarantee

Outline

- Comparison with current DIFC systems
- Laminar: programming model
    - Design: PL + OS techniques
    - Security regions
- Case studies and evaluation
- Summary

Example: calendar

Pseudo code to find a common meeting time for Alice and Bob



Calendar	Monday	Tuesday
Alice	Watch game	Office work
Bob	Free	Meet doctor

```
Calendar cal;  // has label {Alice, Bob}

secure(new Label(Alice, Bob)){
    Calendar a = readFile(“alice.cal”);
    Calendar b = readFile(“bob.cal”);
    cal.addDates(a, b);
    Date d = cal.findMeeting();
… }
catch(..){}
```

Can read data of Alice and Bob.

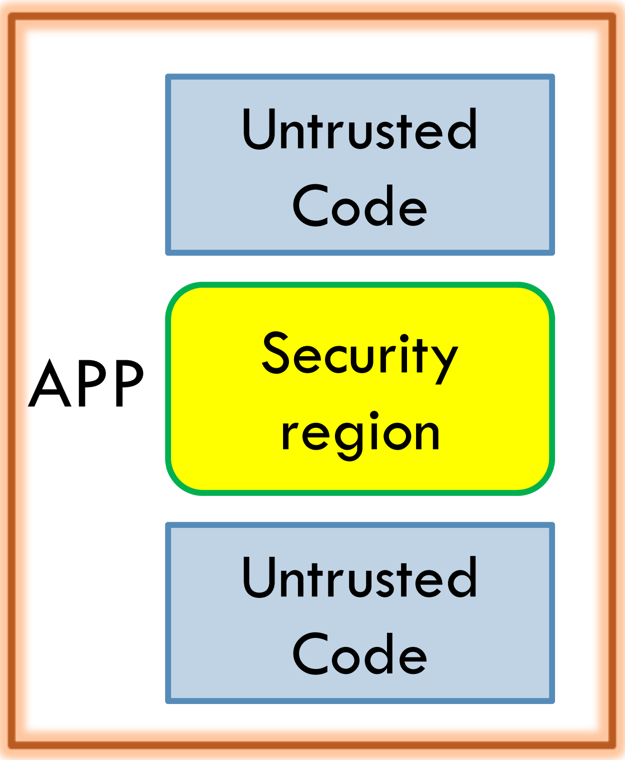
Read data of Alice and Bob.

Add to common calendar

Find common meeting time

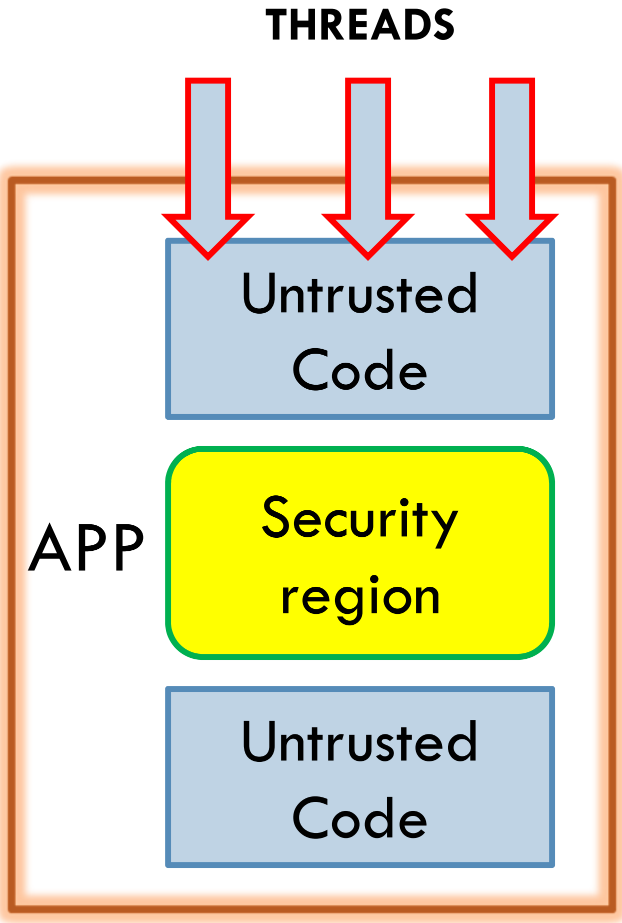
This code has been simplified to help explanation. Refer to the paper for exact syntax.

Security regions for programming ease



- Easier to add security policies
  - Wrap code that touches sensitive data inside security region
  - Hypothesis: only small portions of code and data are security sensitive
- Simplifies auditing

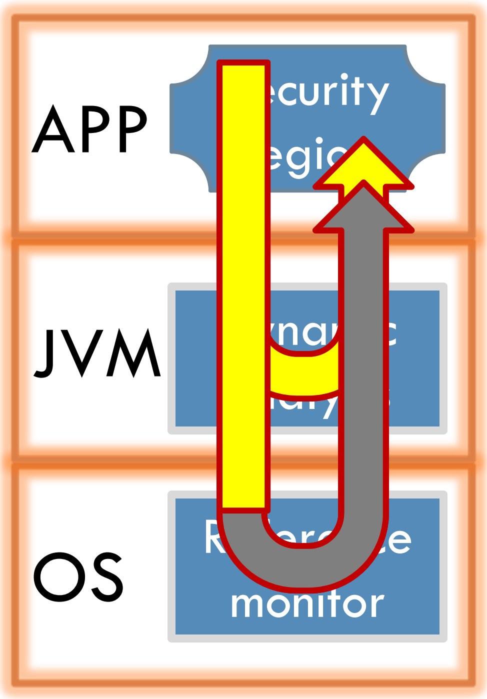
Threads and security regions



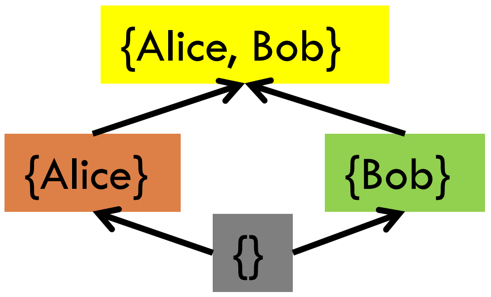
❑ Threads execute the application code

❑ On entering, threads get the labels and privileges of the security region

Supporting security regions: JVM+OS



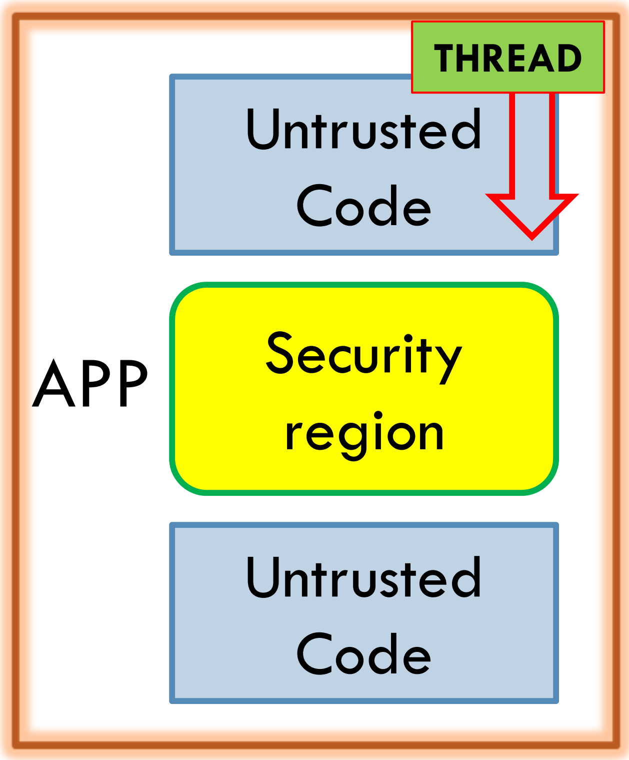
```
Calendar cal;  // has label {Alice, Bob}

secure(new Label(Alice, Bob)){
      Calendar a = readFile("alice.cal");
      Calendar b = readFile("bob.cal");
      cal.addDates(a, b);
      Date d = cal.findMeeting();
… }
catch(..){}
```



Labeling application data

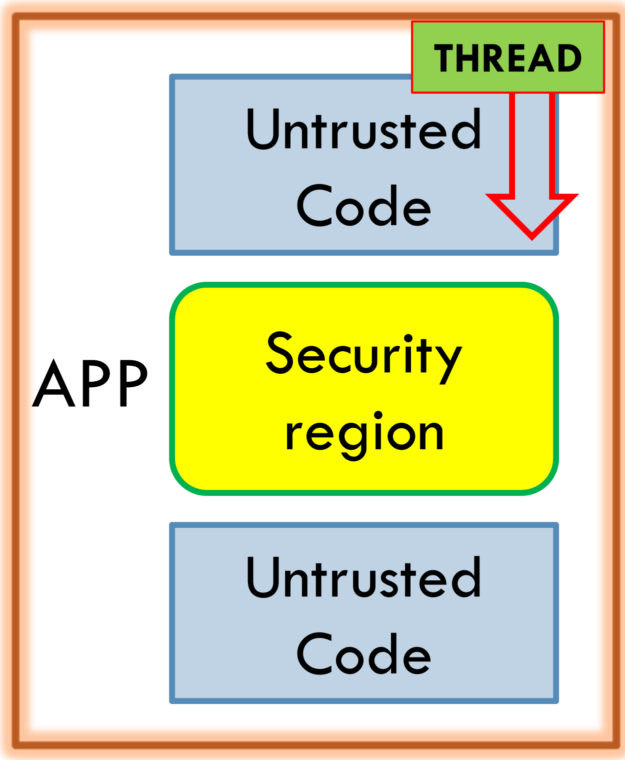
- JVM allocates labeled objects from a separate heap space
  - Efficient checks on whether an object is labeled
  - Object header points to secrecy and integrity labels
- Locals and statics are not labeled
  - Restricted use inside and outside security regions
  - Prevents illegal information flow
- We are extending our implementation to support labeled statics

Security regions for efficiency



- Limits the amount of work done by the VM to enforce DIFC
- Prevent access to labeled objects outside security regions
- Use read/write barriers
- Perform efficient address range checks on objects

Checks outside a security region

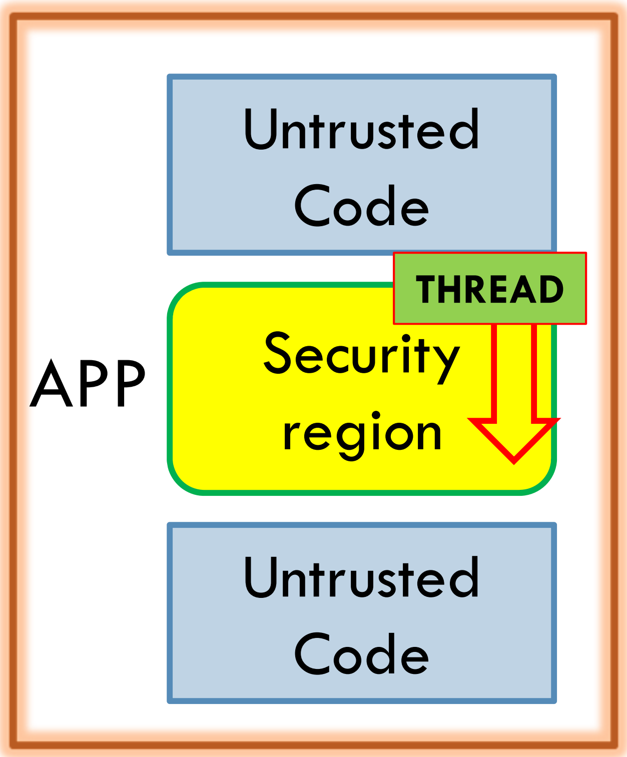


```
Label credentials = new Label (Alice, Bob);
Calendar cal;  // has label {Alice, Bob}

secure(credentials){
    ...
    cal.addDates(a, b);
    Date d = cal.findMeeting();
... }
catch(..){}

Date d= cal.getMeetTime();
```

✗

Labeled object read outside the security region

Checks inside a security region



- Mandatory DIFC checks inside security regions

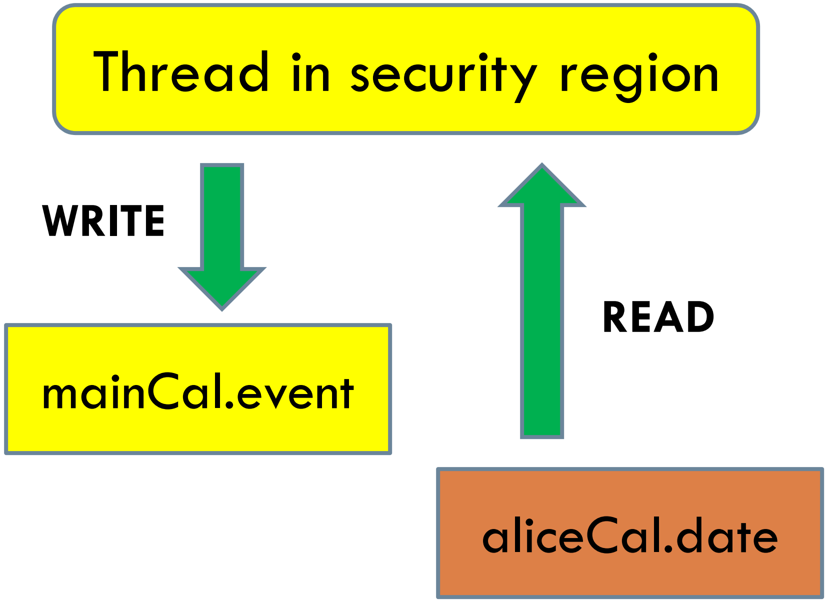
- Secrecy rule
  - Cannot read *more* secret
  - Cannot write to *less* secret

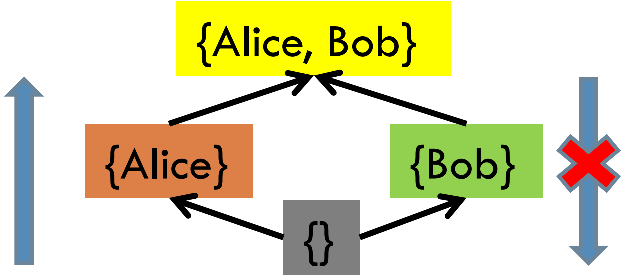
- Integrity rule
  - Cannot read *less* trusted
  - Cannot write to *more* trusted

Checks inside a security region

```
Label credentials = new Label (Alice, Bob);
Calendar mainCal;  // has label {Alice, Bob}
Calendar aliceCal; //has label {Alice}

secure(credentials){
    ...
    mainCal.event = aliceCal.date;

... }
catch(..){}
```

Information flow





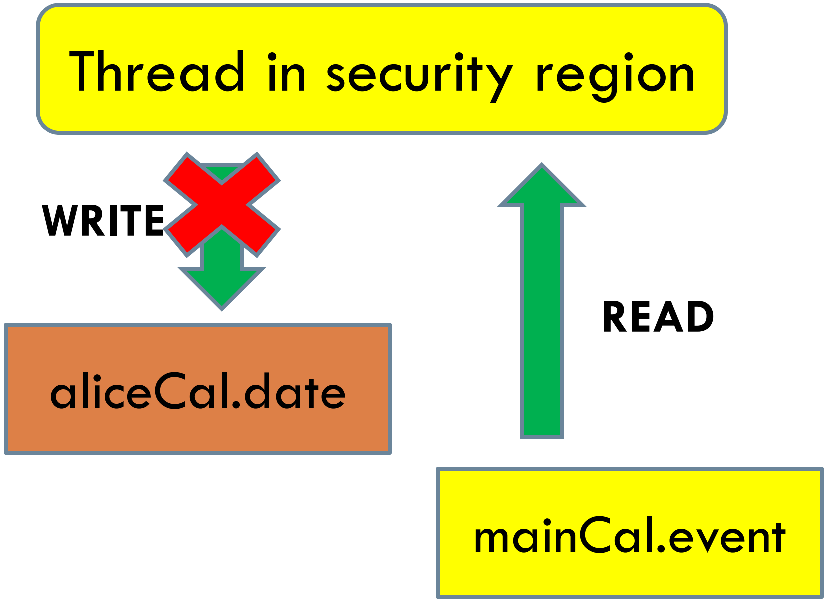
Information flow in a lattice

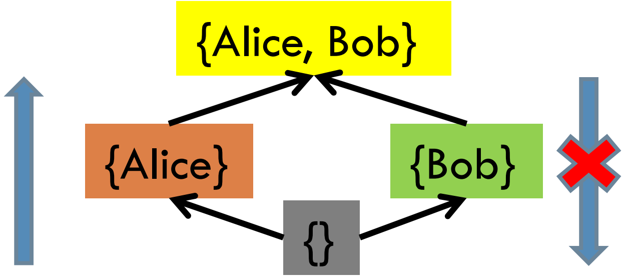
Checks inside a security region

```
Label credentials = new Label (Alice, Bob);
Calendar mainCal;  // has label {Alice, Bob}
Calendar aliceCal; //has label {Alice}

secure(credentials){
    …
    aliceCal.date = mainCal.event ;
```

Information flow

```
… }
catch(..){}
```



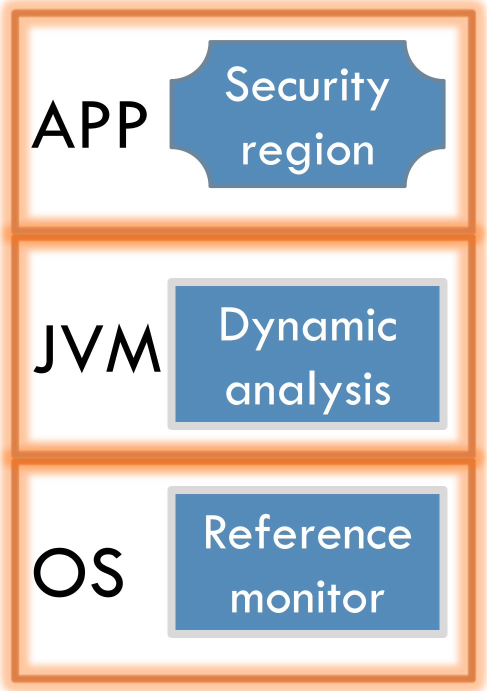


Information flow in a lattice

Nested security regions

- Laminar allows nesting of security regions
- For nesting, the parent security region should have the correct privileges to initialize the child security region
  - Natural hierarchical semantics
- More details are present in the paper

Supporting security regions: OS



- OS acts as a repository for labels
  - New labels can be allocated using a system call
- Labels stored in security fields of the kernel objects
- Before each resource access, the reference monitor performs DIFC checks
  - E.g. inode permission checks, file access checks

Outline

- Comparison with current DIFC systems
- Laminar: programming model
  - Design: PL + OS techniques
  - Security regions
- Case studies and evaluation
- Summary

Evaluation hypothesis

- Laminar requires modest code changes to retrofit security to applications
    - Less burden on the programmer
- Laminar incurs modest overheads
    - Practical and efficient

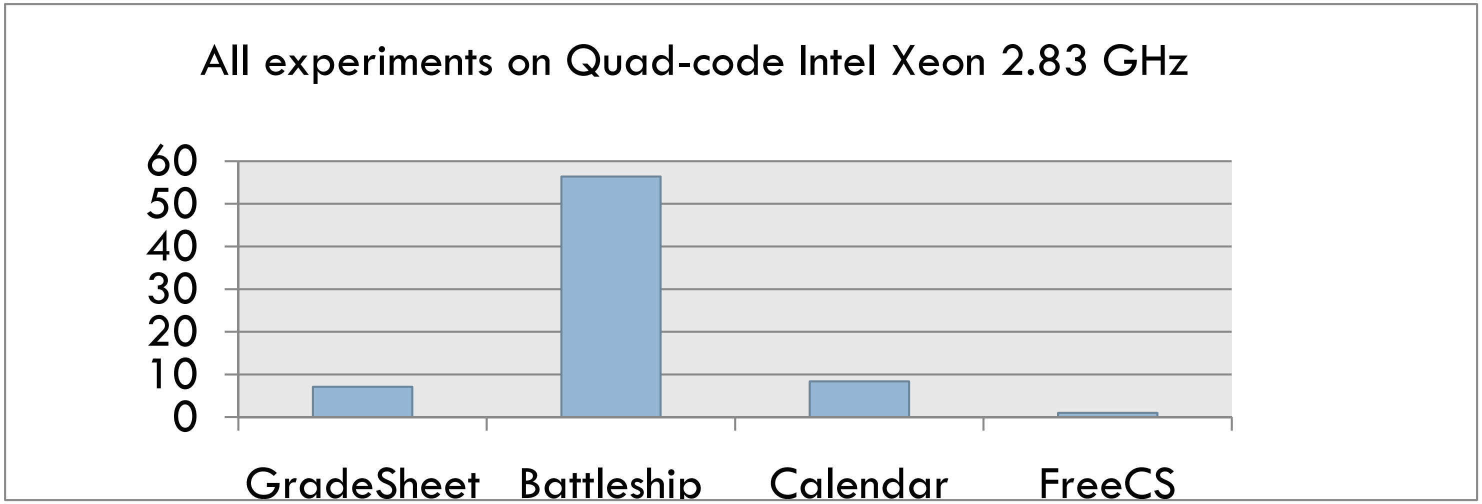
Laminar requires modest changes

Application	LOC	Protected Data	LOC Added
GradeSheet	900	Student grades	92 (10%)
Battleship	1,700	Ship locations	95 (6%)
Calendar	6,200	Schedules	290 (5%)
FreeCS (Chat server)	22,000	Membership properties	1,200 (6%)

≤10% changes

Laminar has modest overheads

- Compared against unmodified applications running on unmodified JVM and OS
- Overheads range from 1% to 54%
- IO disabled to prevent masking effect
  - Lower overheads expected in real deployment


All experiments on Quad-code Intel Xeon 2.83 GHz

Related Work

- IFC and lattice model
  - Lattice Model[Denning'76], Biba'77, Bell-LaPadula'73
- Language level DIFC
  - Jif[Myers'97], FlowCaml[Simonet'03], Swift[Chong'07]
- OS based DIFC
  - Asbestos[Efstathopoulos'05], HiStar[Zeldovich'06], Flume[Krohn'07], DStar[Zeldovich'08]

Summary

Current DIFC systems fall short of enforcing comprehensive DIFC policies

Laminar solves this by introducing security regions and integrating PL + OS mechanisms

Laminar provides fine-grained DIFC, and yet has low overheads

Thank you!

Current DIFC systems fall short of enforcing comprehensive DIFC policies

Laminar solves this by introducing security regions and integrating PL + OS mechanisms

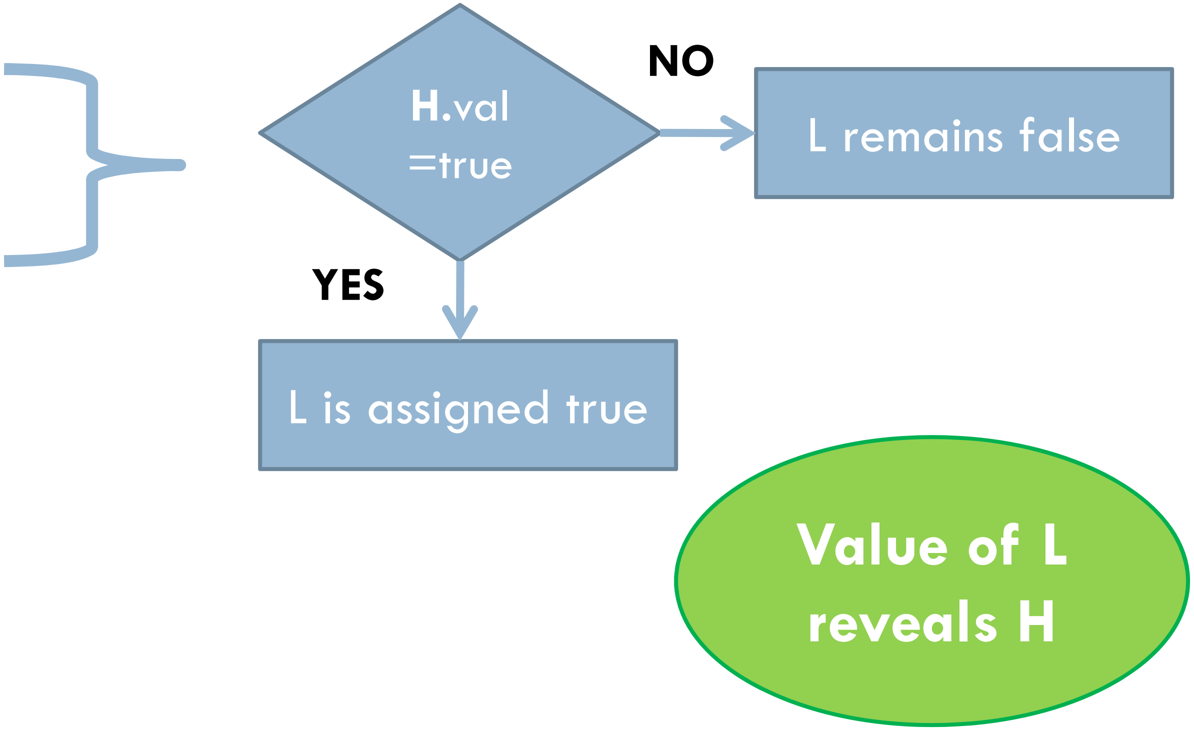
Laminar provides fine-grained DIFC, and yet has low overheads

BACKUP SLIDES !

Implicit information flow

```
// H has label {secret}
// L  has label {}
L.val =  false;

if(H.val)
     L.val = true;
```

H is secret

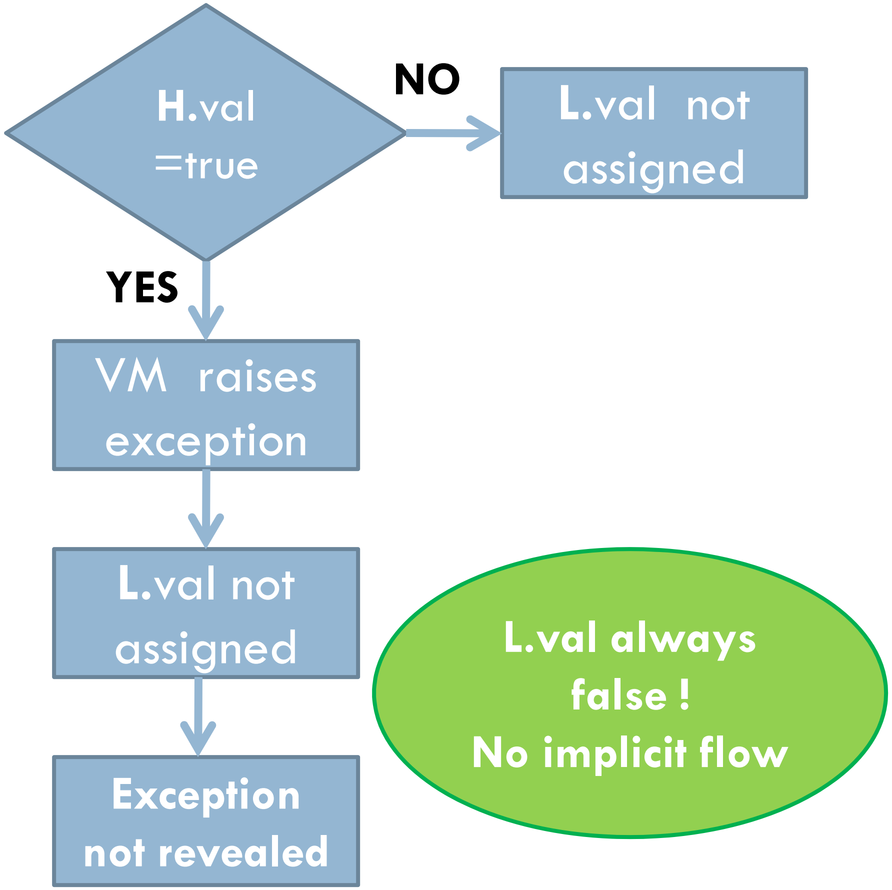


Value of L reveals H

Handling implicit information flows

```
// H has label {secret}
// L  has label {}
L.val =  false;
secure(credentials){
   if(H.val)
         L.val = true;
} catch(…) {
}
```

Mandatory catch block.
Executes with same labels as the security region



L.val always false !
No implicit flow