Seminar on Secure Software

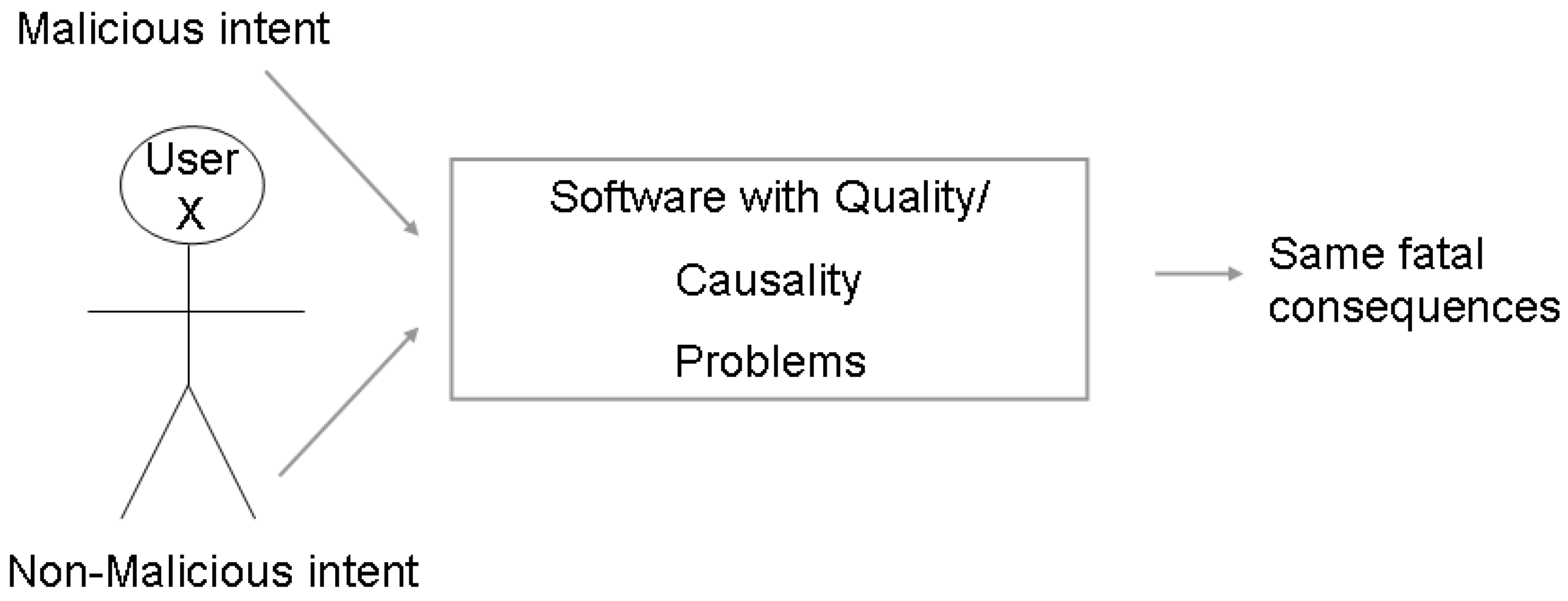
Agenda

1. Security as a Subset of Safety
2. Problem View: Beyond Attacks
3. Critical Trends and Developments
4. Root Causes
5. Architectures for Secure Software
6. Application Analysis
7. Modeling the Flow of Authority
8. Resources

SECTION 1

Theory: Security as a Subset of Safety

Functional Safety vs. System Security – a Real Difference?



Malicious intent

User X

Software with Quality/

Causality

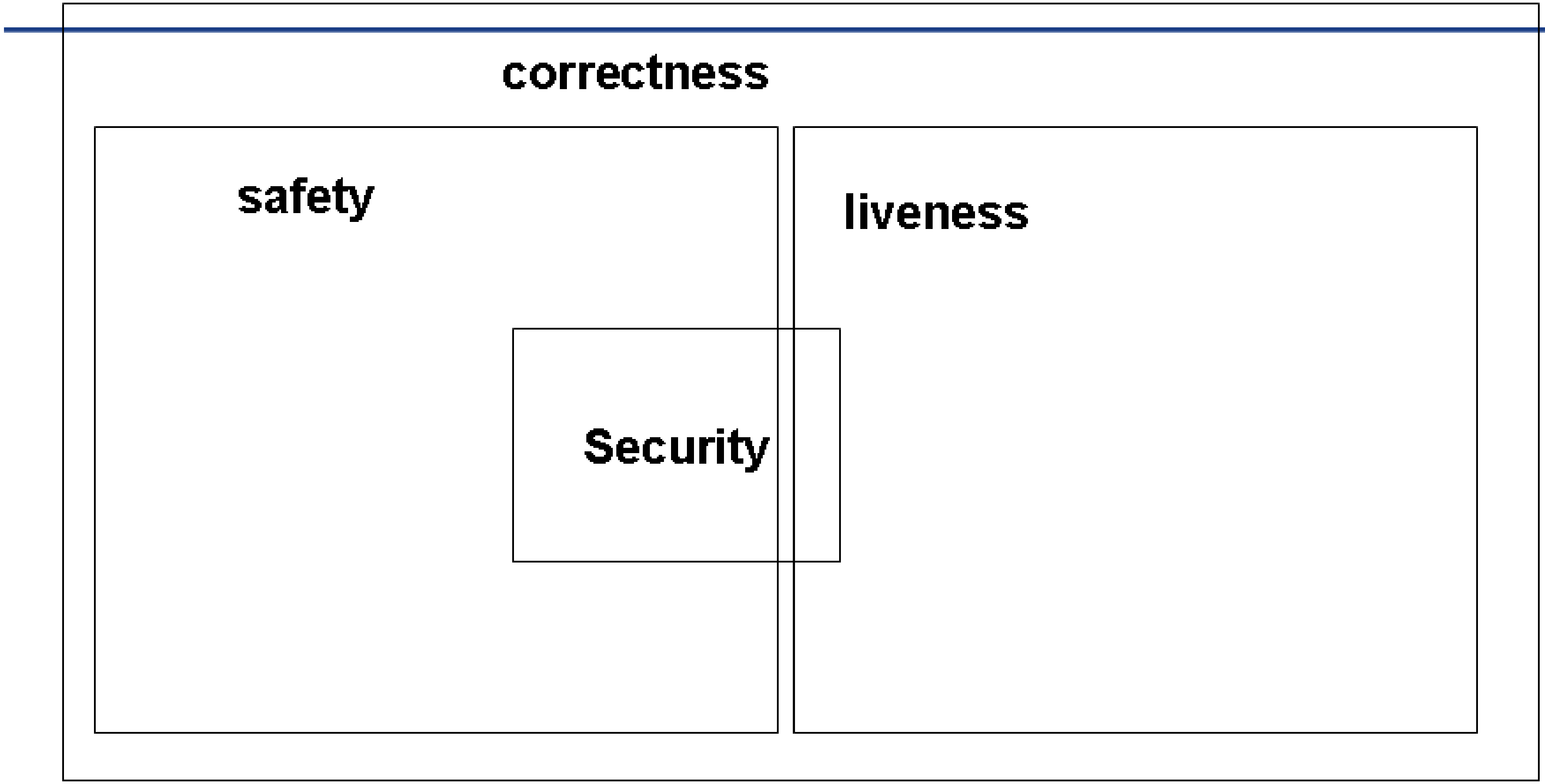
Problems

Same fatal consequences

Non-Malicious intent

Software quality problems are both safety AND security problems as we often won‘t be able to distinguish the intents in the consequences. (see Functional Safety Standard: IEC 61508)

Liveness and Safety



correctness

safety

liveness

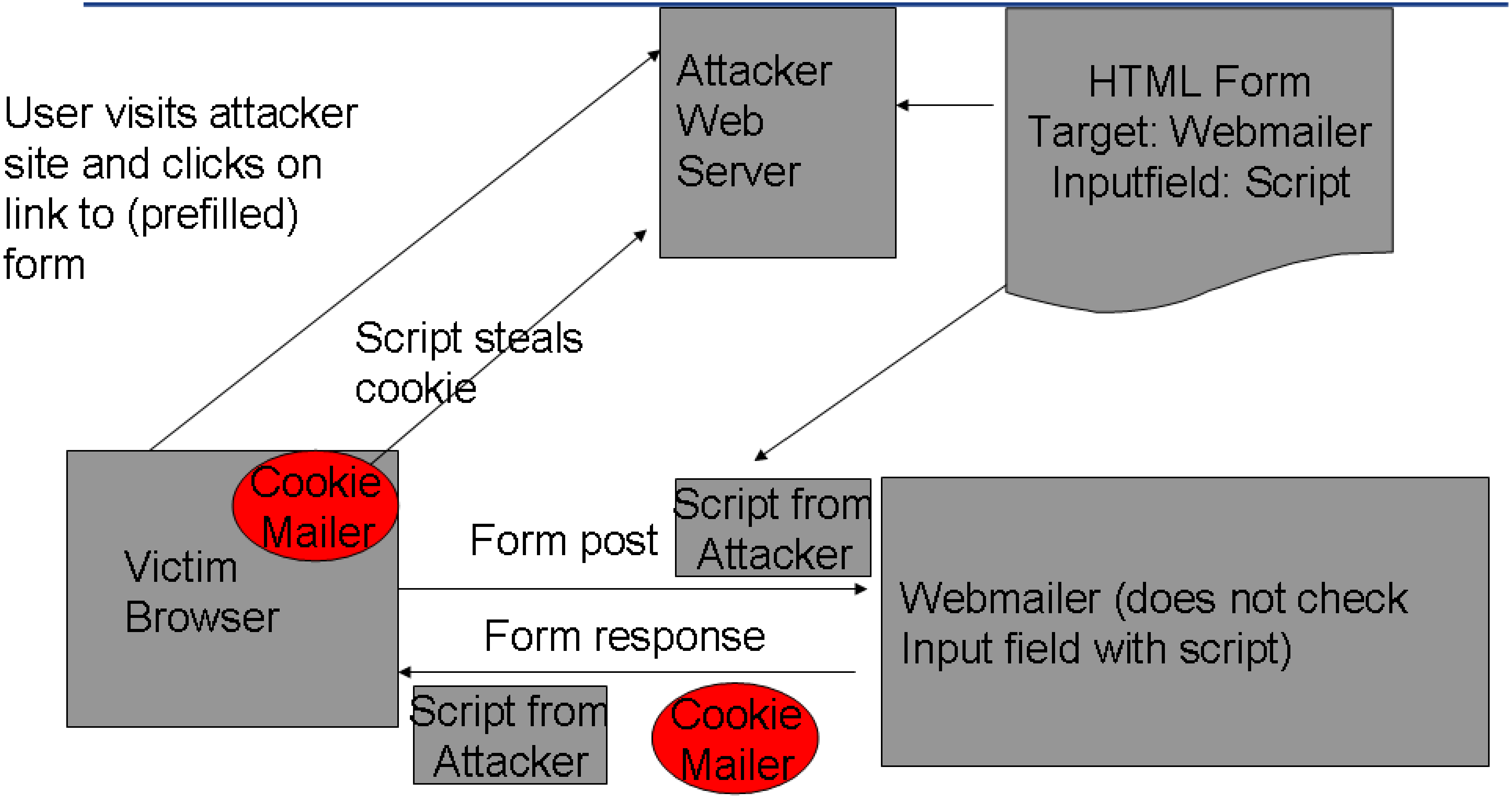
Security

The correctness of a system consists of safety (expressed by things that should NOT happen) and liveness (expressed as things that SHOULD happen). Security issues can be expressed as both with the addition of malicious intent.

SECTION 2

Problem View: Beyond Attacks

Vulnerabilities (1) : Cross-Site Scripting



User visits attacker site and clicks on link to (prefilled) form

Attacker Web Server

HTML Form
Target: Webmailer
Inputfield: Script

Script steals cookie

Victim Browser

Cookie Mailer

Form post

Script from Attacker

Webmailer (does not check Input field with script)

Form response

Script from Attacker

Cookie Mailer

Cross-Site Scripting – one of many cases of bad input/output validation in applications. Where are the frameworks to help developers? How many ways are there to express a „<„ in unicode? Is this problem really solvable? Think about the relation between input and interpreter – one man‘s trash is another man‘s treasure! **And exactly WHY is XSS so dangerous?**

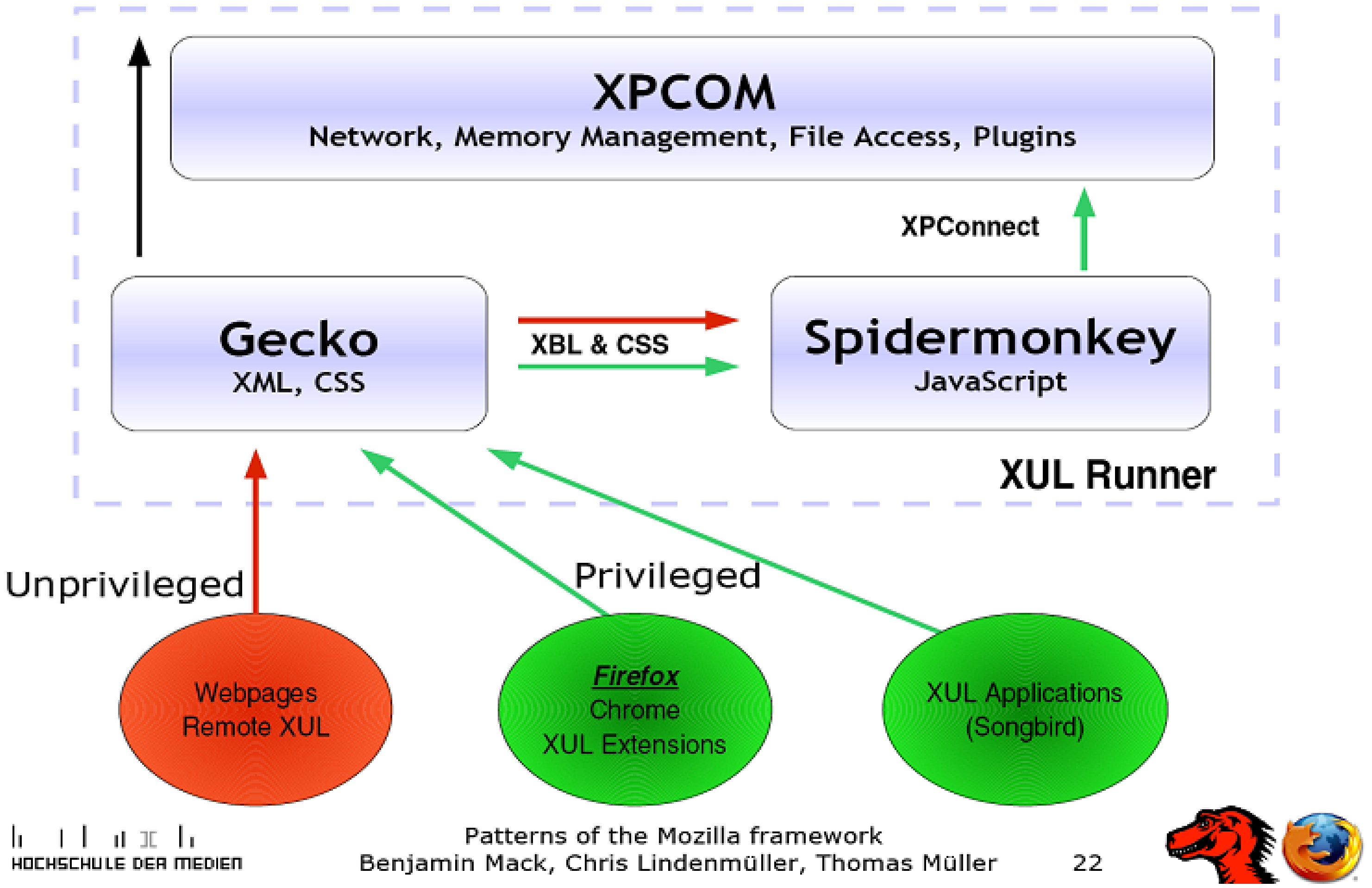
Vulnerabilities (2) : Buffer-Overflow

Our „aaaaaaaa..“ input from the keyboard is now the address where the next instruction should be read by the CPU. Now we know how to point the CPU to code we placed on the stack

```
Exception: STATUS_ACCESS_VIOLATION at eip=61616161
eax=00000012 ebx=00000004 ecx=610E3038 edx=00000000 esi=004010AE
edi=610E21A0
ebp=61616161 esp=0022EF08
program=D:\kriha\security\bufferoverflow\over.exe, pid 720, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
  90087 [main] over 720 handle_exceptions: Exception:
STATUS_ACCESS_VIOLATION
 104452 [main] over 720 handle_exceptions: Error while dumping state
(probably corrupted stack)
```

A program crash is a way into the system! But the real quality problem is much deeper: Stick a finger in some code and figure out what you can do from there. **What functions can you reach from any point in code? Who‘s failure is that?**

Vulnerabilities (3) : Dangerous Extensions

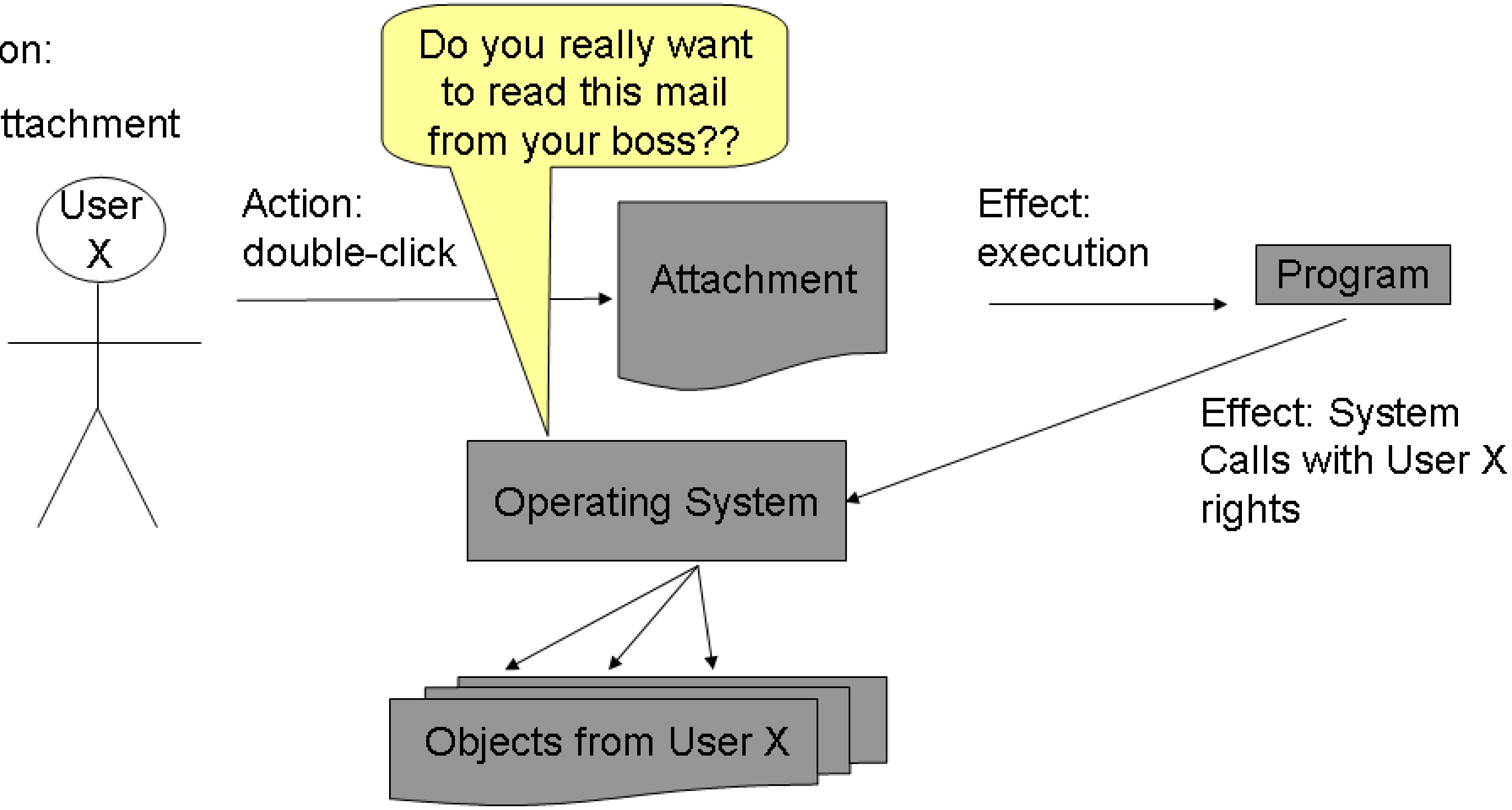


Extensions are a necessity nowadays (eclipse, 3dsmax, firefox) get most of their functions through plug-ins. Linux and XP use the same principles for the OS kernel. But is it OK that every extension can take over the application or system? **Will a simple privileged vs. Unprivileged mode do?**

Vulnerabilities (4) : Virus/Trojan Horse

Intention:

read attachment



One thing to remember when a virus or trojan ruins your computer: The operating system WORKS AS SPECIFIED in this case. So it must be your fault, or?

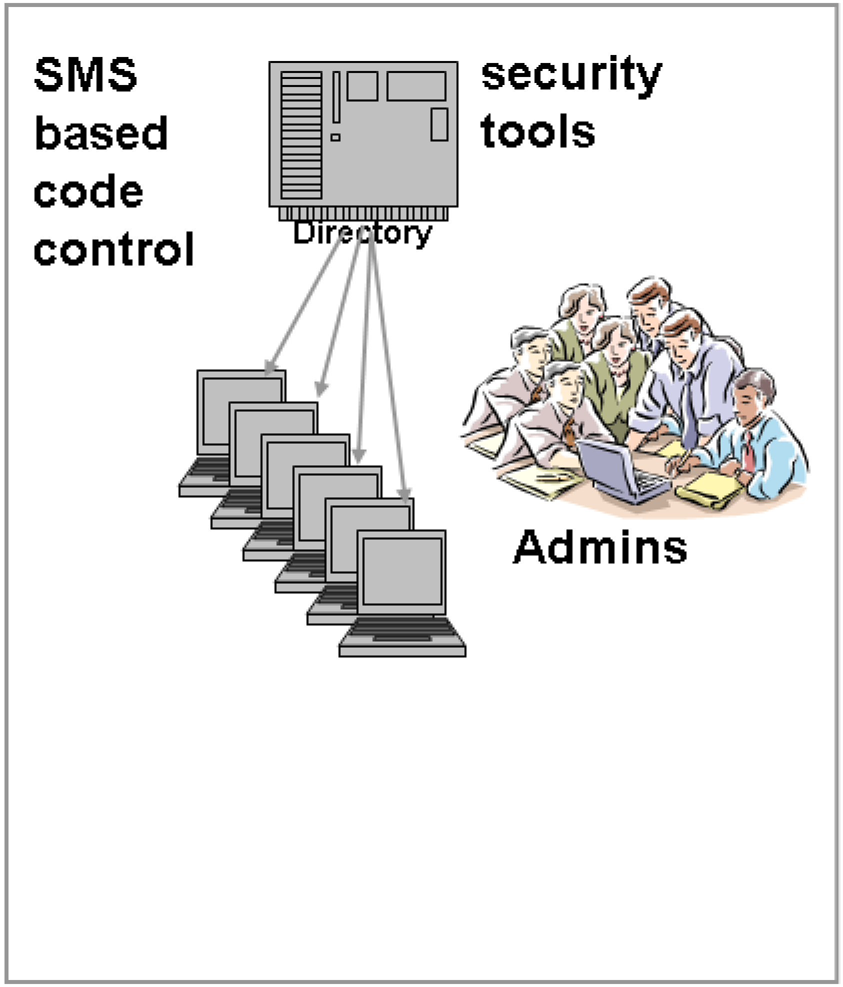
On getting used to something: do you hear talking regular people about which tools they need to drive their cars? **Why then talk regular people about firewalls, virus scanners, privacy guards etc.?**

SECTION 3

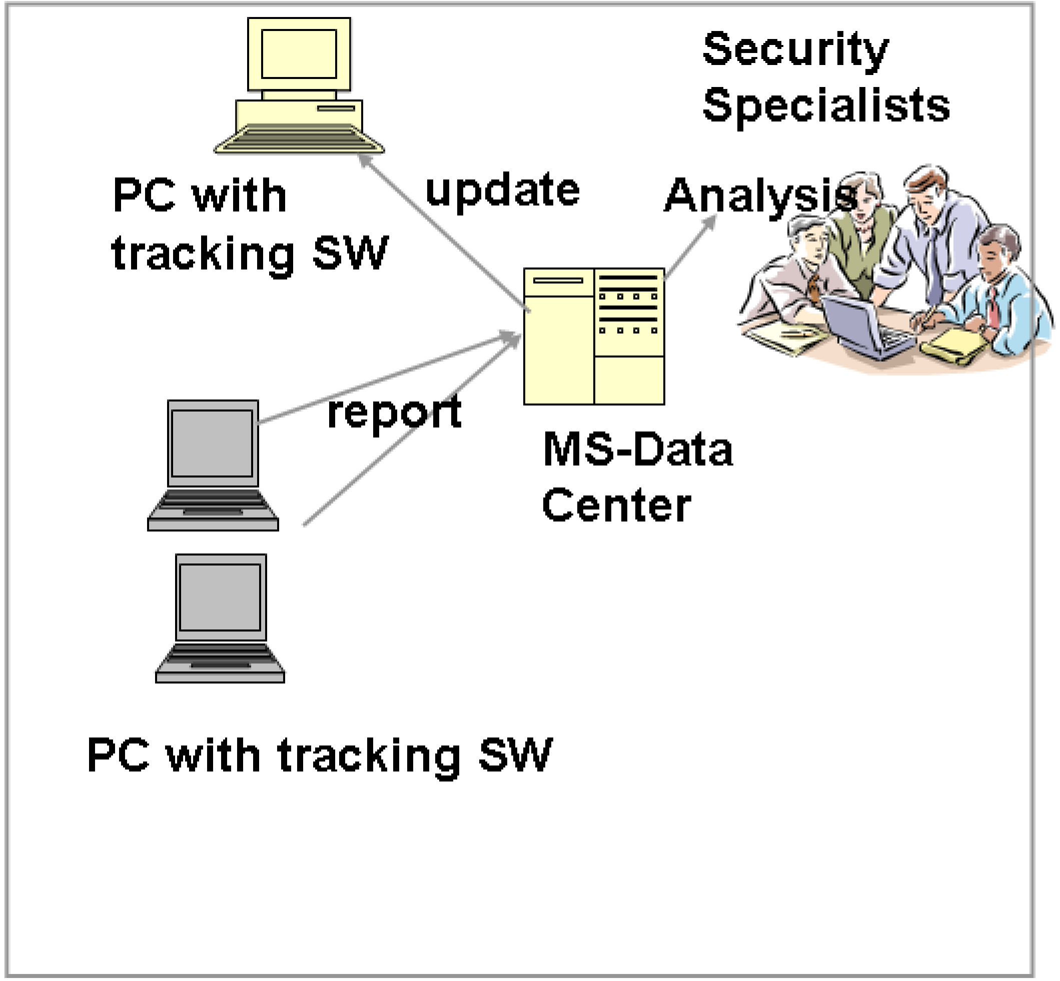
Critical Trends and Developments

Giving up on Platform Security?

office



home



Bill Gates is wrong: not the Internet is unsafe and dangerous – it is his platform. And the solution will not be global data centers controlling the home computing platforms. **Is there any reason to assume that software is inherently unsafe?**

Sandboxes and the „AllPermission“ Problem

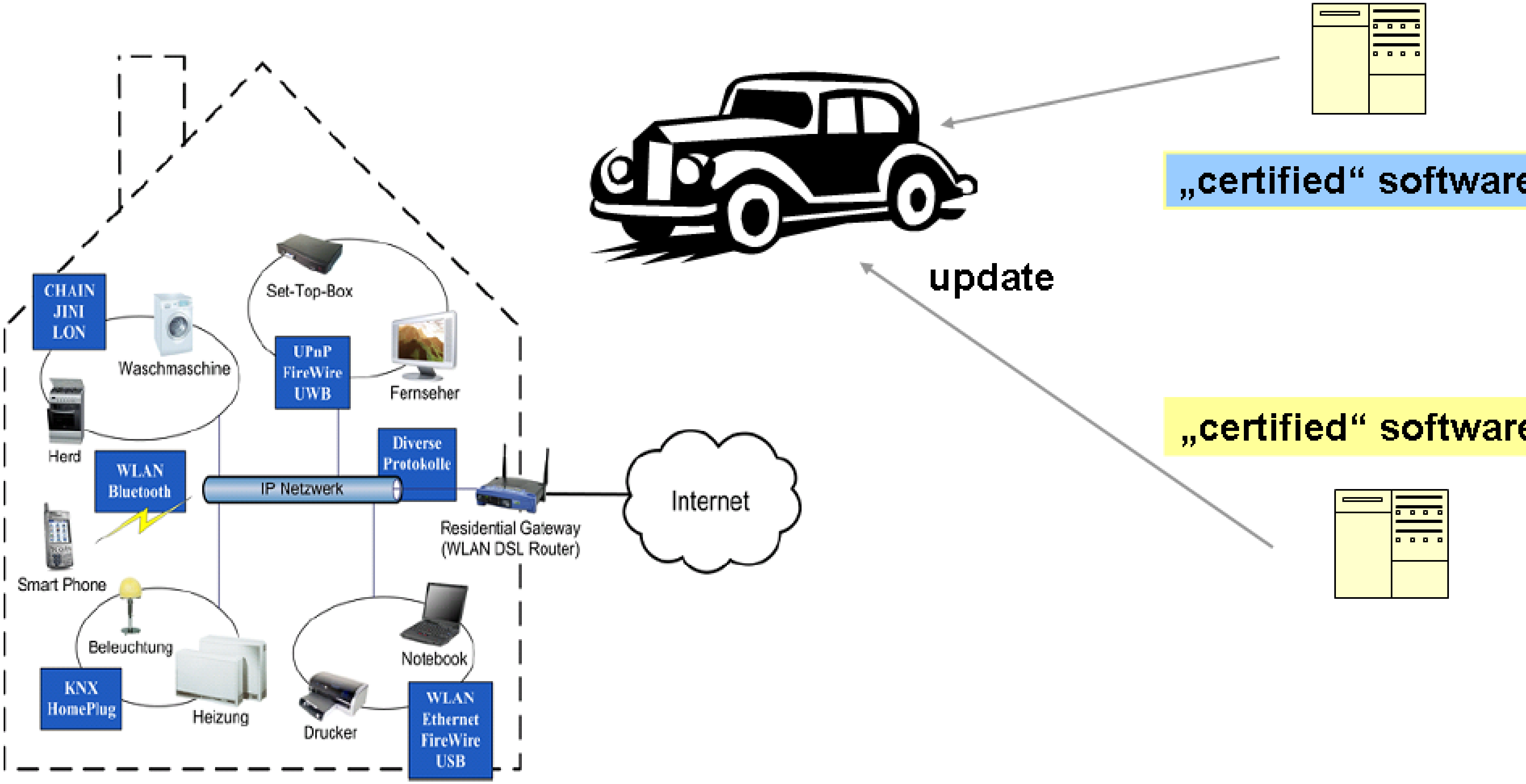
-Java 2 Security „AllPermission“

- .Net „fully authorized“

-- Symbian OS „*.* Permission“

Many platforms allow restrictions of authority on different levels of granularity (objects, assemblies, dlls). Why are those mechanisms almost never used? **Could it be the consequences for architecture and design?**

Ubiquitous computing



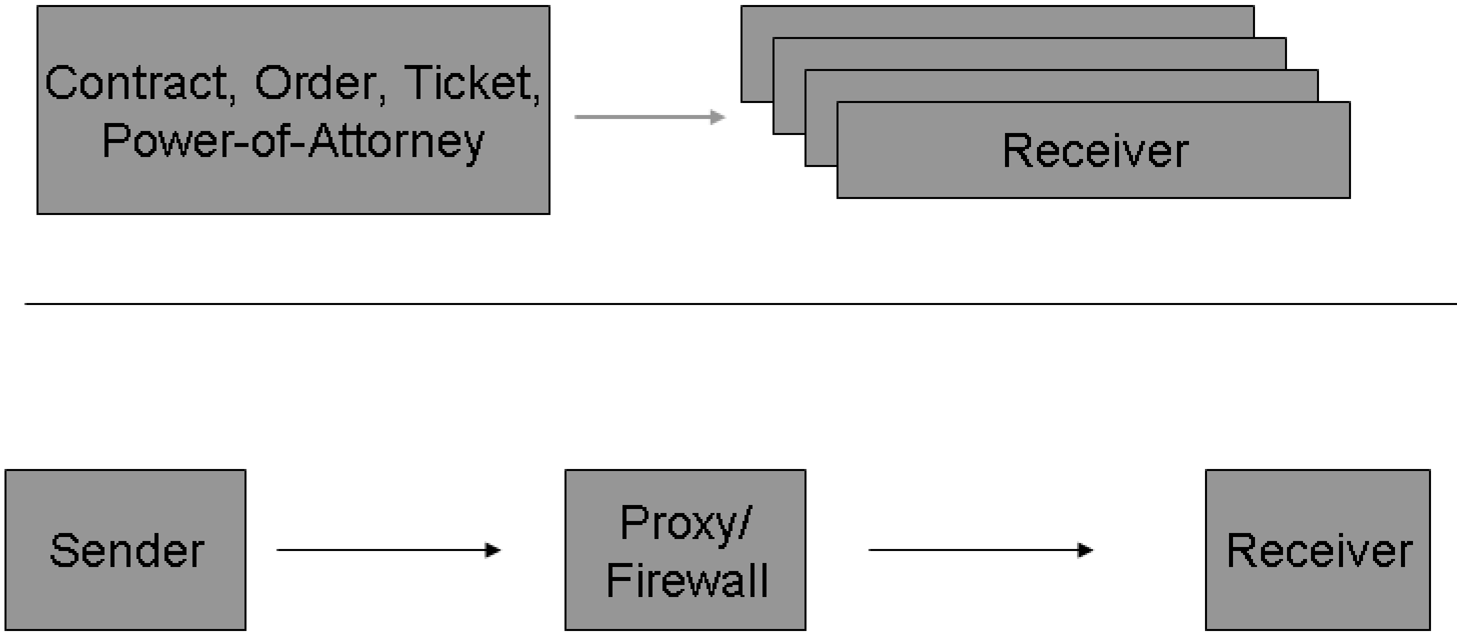
„certified“ software

update

„certified“ software

Cars, personal appliances, shops and transport agents etc. will all communicate with each other. How do we manage our privacy and intentions in this context? Autonomous agents need power and independence to do their job – and **there is no margin for error or security holes.**

Beyond Infrastructure Security: Distributed Security



Signed Messages travelling across intermediates

Channel based security infrastructure

We will go from channel based security to a cryptographically based form of communication that is modelled after real world security in our society. This will **decouple infrastructure and security better.**

SECTION 4

Root Causes

Software Defects that threaten Safety

A common, navigable filesystem with ambient authority

Dangerous accounts, single audit and log features

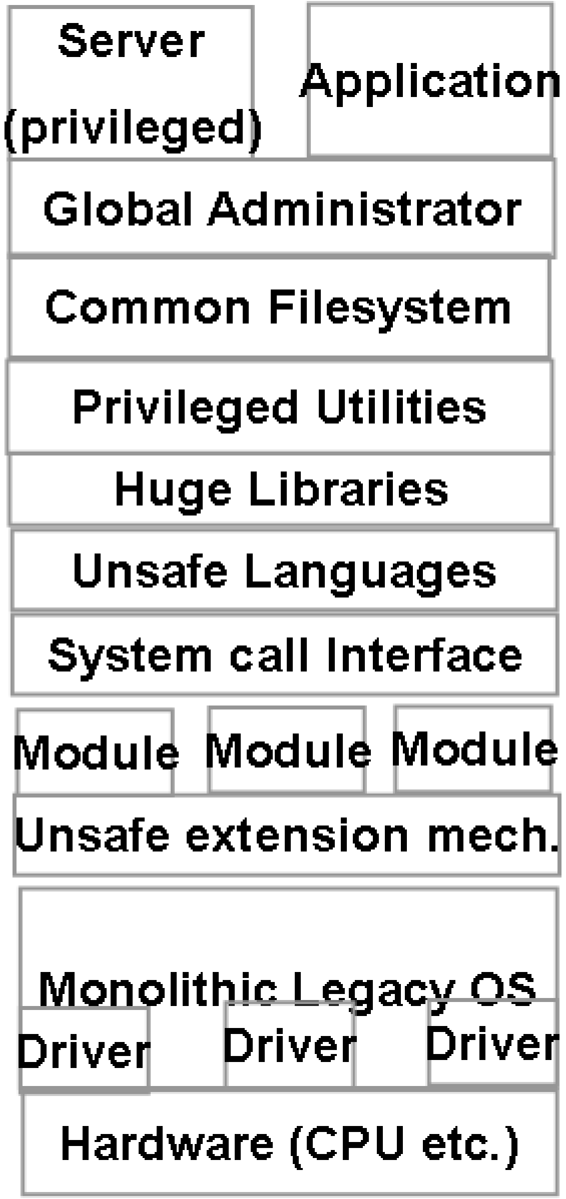
Tons of unsafe but privileged scripts and utilities (setUid)

>300 complex system calls

Countless dynamically loadable modules

>100.000 drivers for windows

Huge TCB, 2 modes only



Server (privileged)

Application

Global Administrator

Common Filesystem

Privileged Utilities

Huge Libraries

Unsafe Languages

System call Interface

Module

Module

Module

Unsafe extension mech.

Monolithic Legacy OS

Driver

Driver

Driver

Hardware (CPU etc.)

Same runtime for all applications

Cycle stealing applications create a problem for near-realtime multimedia applications

Lots of unverified system libraries with memory leaks etc.

Incomplete quota administration (liveness problems)

Attacks on random number generation

Unsafe languages (memory)

Unsafe extension mechanisms

Covered channels (cache, bios, CPU)

Some Reasons for Insecure Software

-Ambient Authority makes errors and attacks fatal (No Loader Isolation etc.)

-Missing frameworks for input validation based on a definition of the application language

-Extension concepts that do not provide loader isolation

-No authority reduction strategies within applications

-No granular delegation of rights

- Separation of designation from authority (confused deputy)

-A huge dependency on infrastructure security (.NET and J2EE)

-Bad testing approaches (no fuzzers, no automation etc.)

-Bad Shared-state multithreading

-Side-effects, global directories, global navigation, security modes etc.

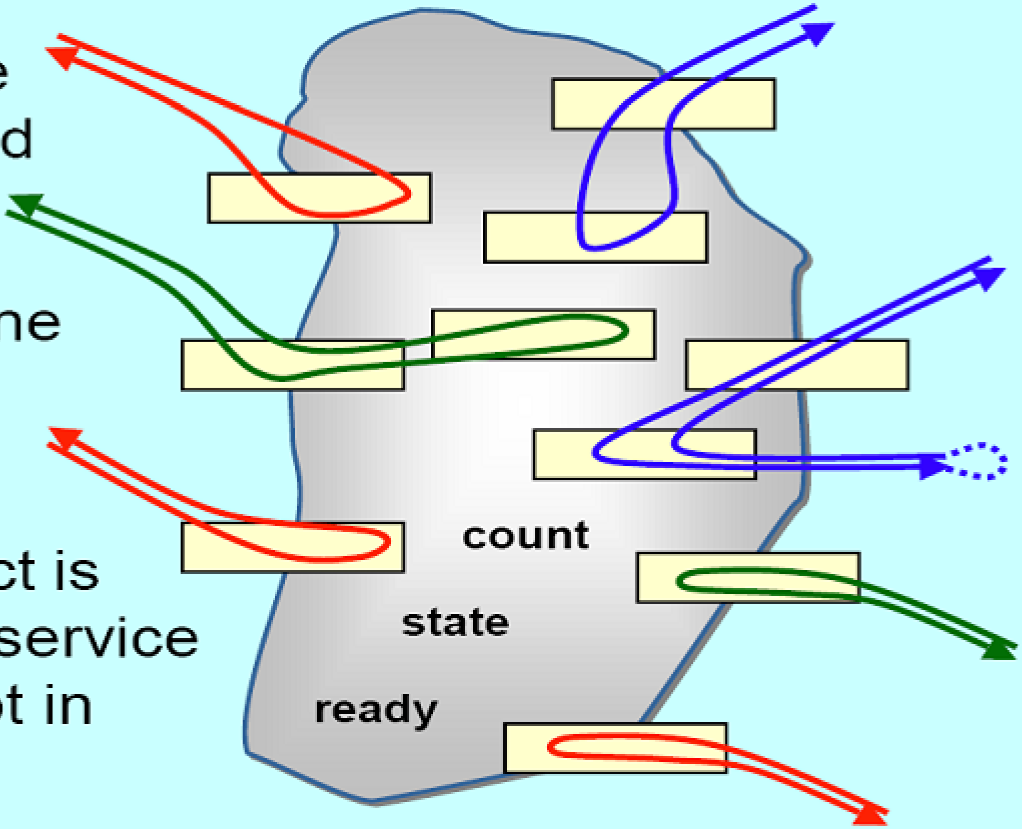
Deploying an application into this environment can take month after month of laborious testing. But how can you be sure that core security concepts (like trust zones, end-to-end security, secrecy etc.) are met and maintained by the software? Automation of tests is a key requirement! Execution of tests must be fully traced.

Example: multi-threading hell

The object is at the mercy of *any* thread that sees it.

Nothing can be done to prevent method invocation ...

… even if the object is not in a fit state to service it. The object is not in control of its life.



Now isn‘t that a funny way to do encapsulation? From P. Welch, a CSP Library for Java Threads. We see not only consistency and liveness problems with shared-state multi-threading but also subtle time-of-check-to-time-of-use (TOC2TOU) security problems.

Example: The designation problem

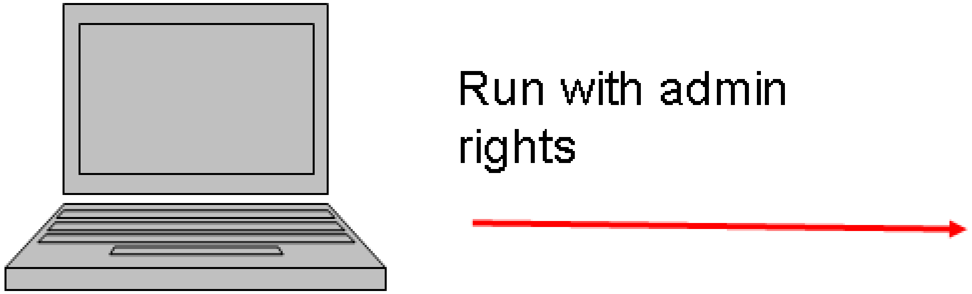
```
Open (char* filename, int mode) // application needs to transform the
                                // symbolic filename into a ressource
```

```
Open (Filedescriptor fd) // application receives an open resource without the
                         // need to perform any rights-related operations
```

An API like this forces the transfer of all authority from the user to the application because it is unclear what file will be opened at runtime. The second API does NOT require ambient authority!

Example: The Installation problem

System (with admin rights)



Run with admin rights

New Software:

- Dll files, config files

- setup.exe

A system with such a software installation process does not need to wonder about software artefacts being scattered througout the system. It is inherently unsafe to call foreign code with admin privileges

SECTION 5

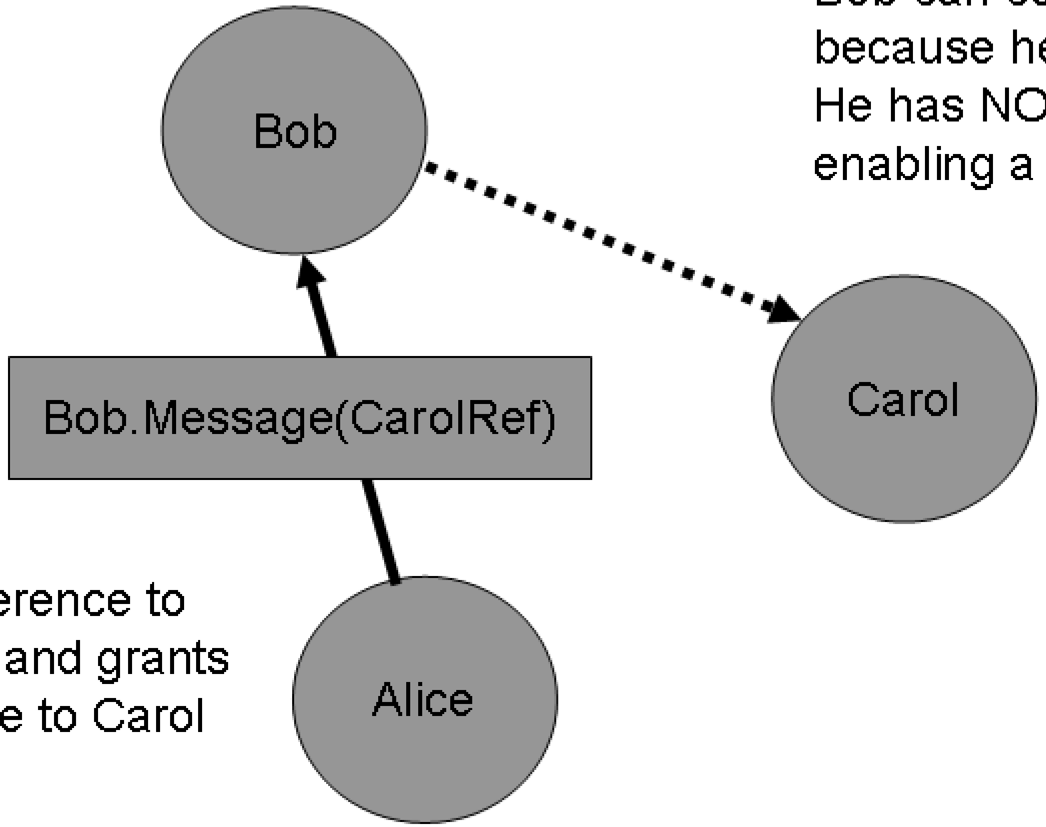
Architectures for Secure Software – or why Security is not an Aspect

Authority Reduction Mechanisms

- Access Control List and Reference Monitors (Operating Systems)
- Call-Trace and Reference Monitors (Java, .NET, distributed objects)
- Multi-Level Security (labels, tagging, tainting)
- Name-Space based isolation (OSGI)
- Object Capabilities (E, Singularity, capability systems)
- cryptographic methods: Contract, ticket, signature, power-of-attorney
- Mode-based Security (e.g. processor rings)
- Zone-based Security (Internet Explorer Zones, Mozilla Chrome)
- Programmatic Security (if-then-else)

The mechanisms are very different along several dimensions: static/dynamic, external/internal, architecture-sensitive vs. Insensitive, data-oriented vs. Code-oriented, type vs. Object based, infrastructure-dependent vs. Independent, granular vs. Global.

Microarchitecture: Object Capabilities

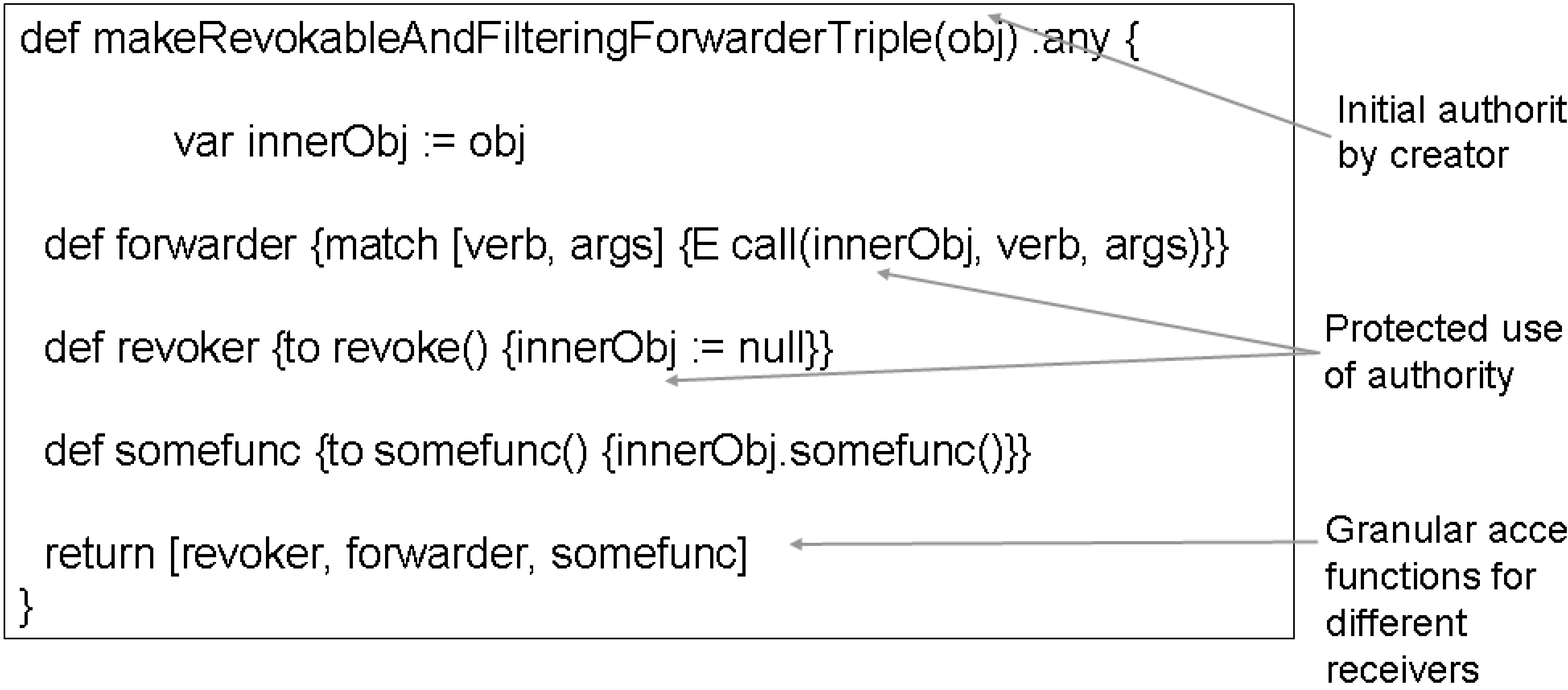


Bob can communicate with Carol because he got a reference to her. He has NO ambient authority enabling a call to Carol

Alice has a reference to Bob and Carol and grants Bob a reference to Carol

Object Capabilities reduce authority in a system: no access without a reference. And references combine access right and access method (designation and authority). They are a superior way to CONSTRAIN effects and are easier to analyze than external permissions. The diagram is called „Granovetter-Diagram“ after the well known sociologist Granovetter).

Microarchitecture: Functional Principles



```
def makeRevokableAndFilteringForwarderTriple(obj) :any {
            var innerObj := obj
  def forwarder {match [verb, args] {E call(innerObj, verb, args)}}
  def revoker {to revoke() {innerObj := null}}
  def somefunc {to somefunc() {innerObj.somefunc()}}
  return [revoker, forwarder, somefunc]
}
```

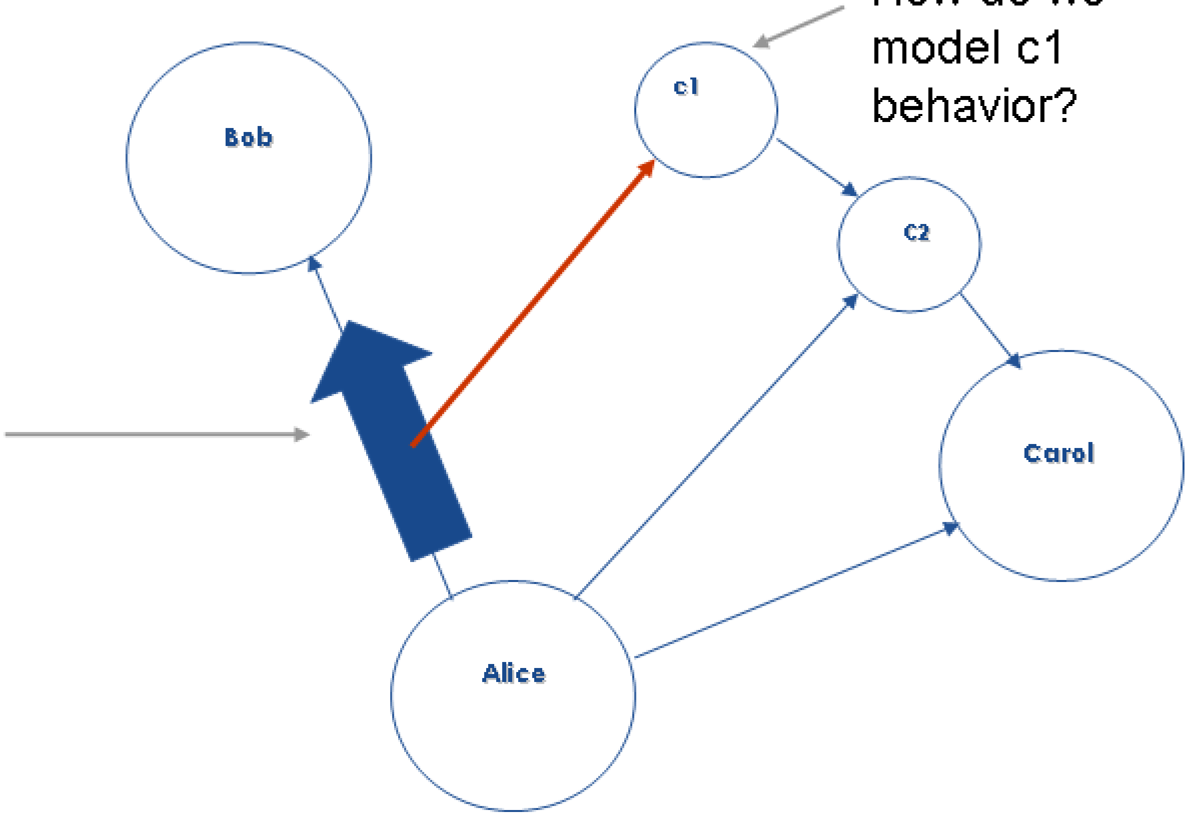
Initial authorit by creator

Protected use of authority

Granular acce functions for different receivers

Higher-order functions and closures have important security properties: they can encapsulate authority and provide granular access to it. And they make authority accessible only while they run (Execution environment). They behave like mathematical functions with respect to predictable operations. Call by value semantics makes life much easier. Creator and caller can have different privileges. Are our students still learning the concepts behind other languages?

Security Components: Security by Behavior

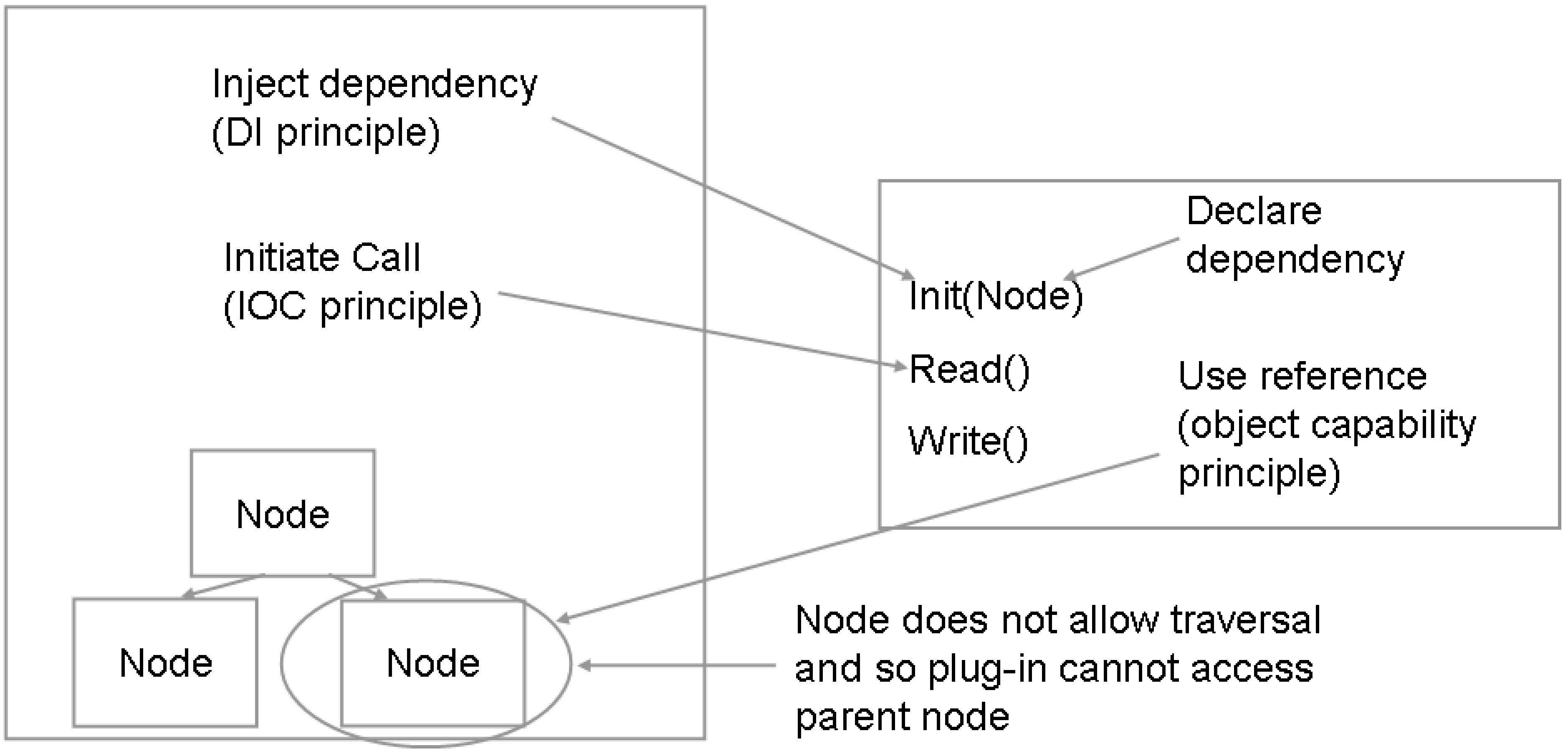


How do we model c1 behavior?

Behavior beyond what simple take/grant systems can express

CT is a „caretaker“ which allows Alice to revoke Bob‘s access to Carol. Modelling security properties becomes much easier once we can include the behavior of code into the security calculations. This allows us to narrow down authority to actual effects.

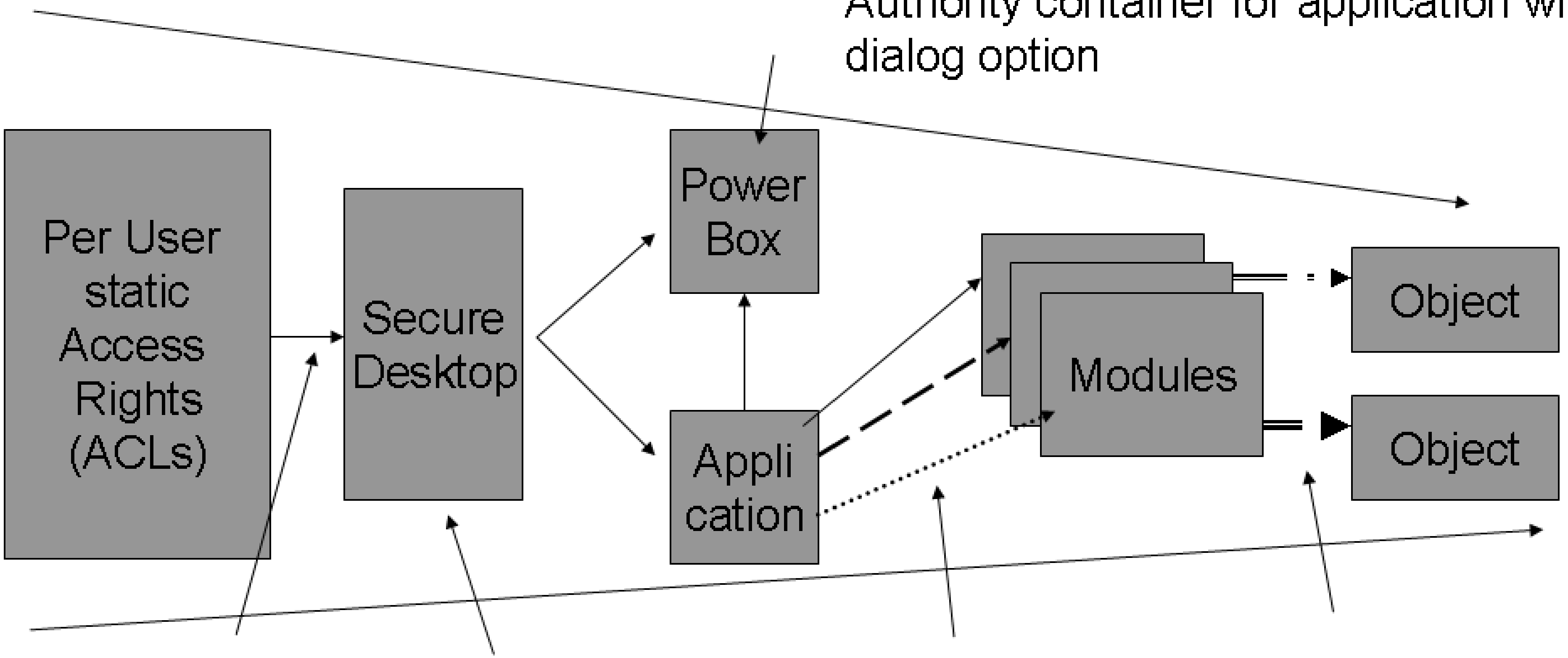
Macro-Architecture: IOC and Virtualization



How do we make extensions safe? How do we achieve complicated business requirements like multi-tenant abilities? The answer is in Inversion-Of-Control architectures combined with strict control over references (no global crap for „flexibility“ reasons…) which effectively virtualizes the plug-in runtime environment

Authority Reduction Architecture

Authority container for application with dialog option



Designation of object and action by User

(trusted path + authority by designation)

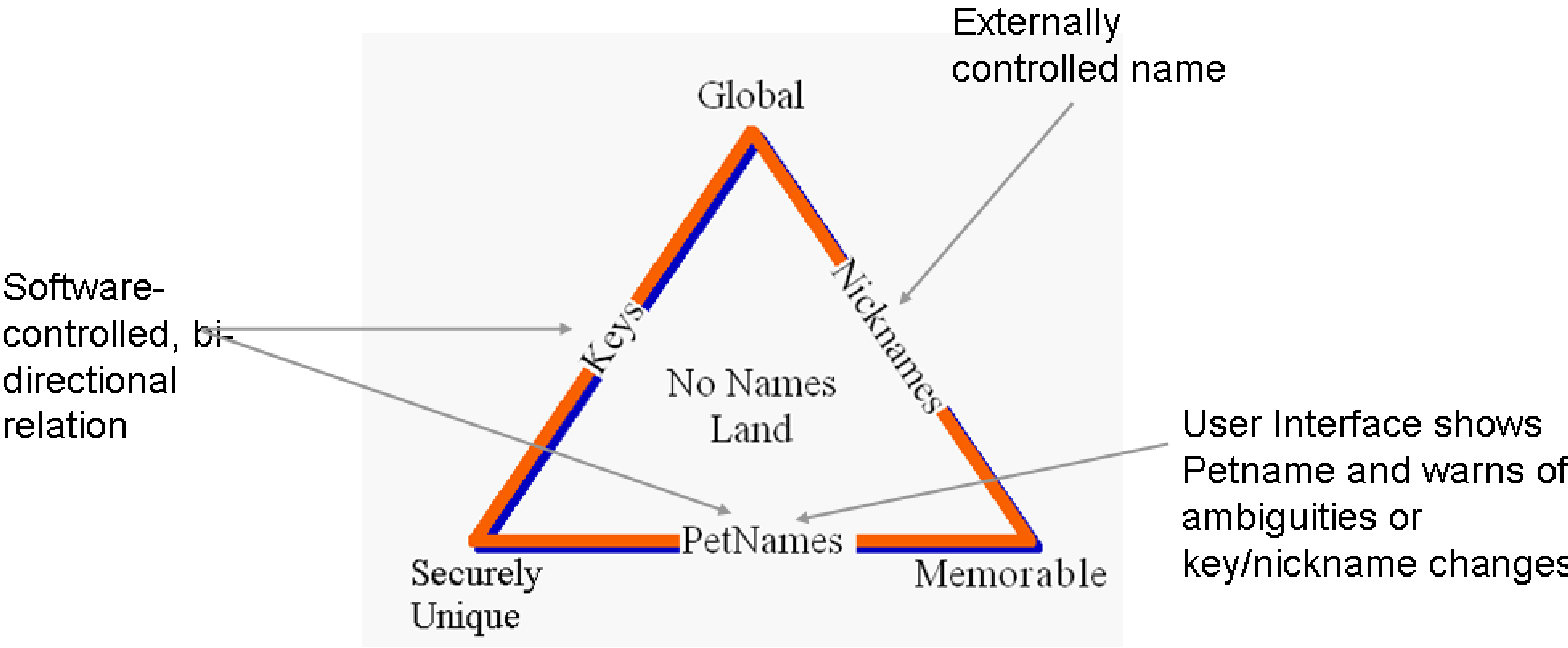
Transformation of names to capabilities and creation of powerbox per application

Granular distribution of authority (capabilities) to program modules

Granular delegation of authority to single objects

We need to narrow authority down from the global rights matrix (ACLs or Access Control Matrix) of a users rights to the minimum authority necessary to exectute a function. Test: try to find how many rights you REALLY need to copy a file!

Semantics and Usability: Petname Systems



Externally controlled name

Software-controlled, bi-directional relation

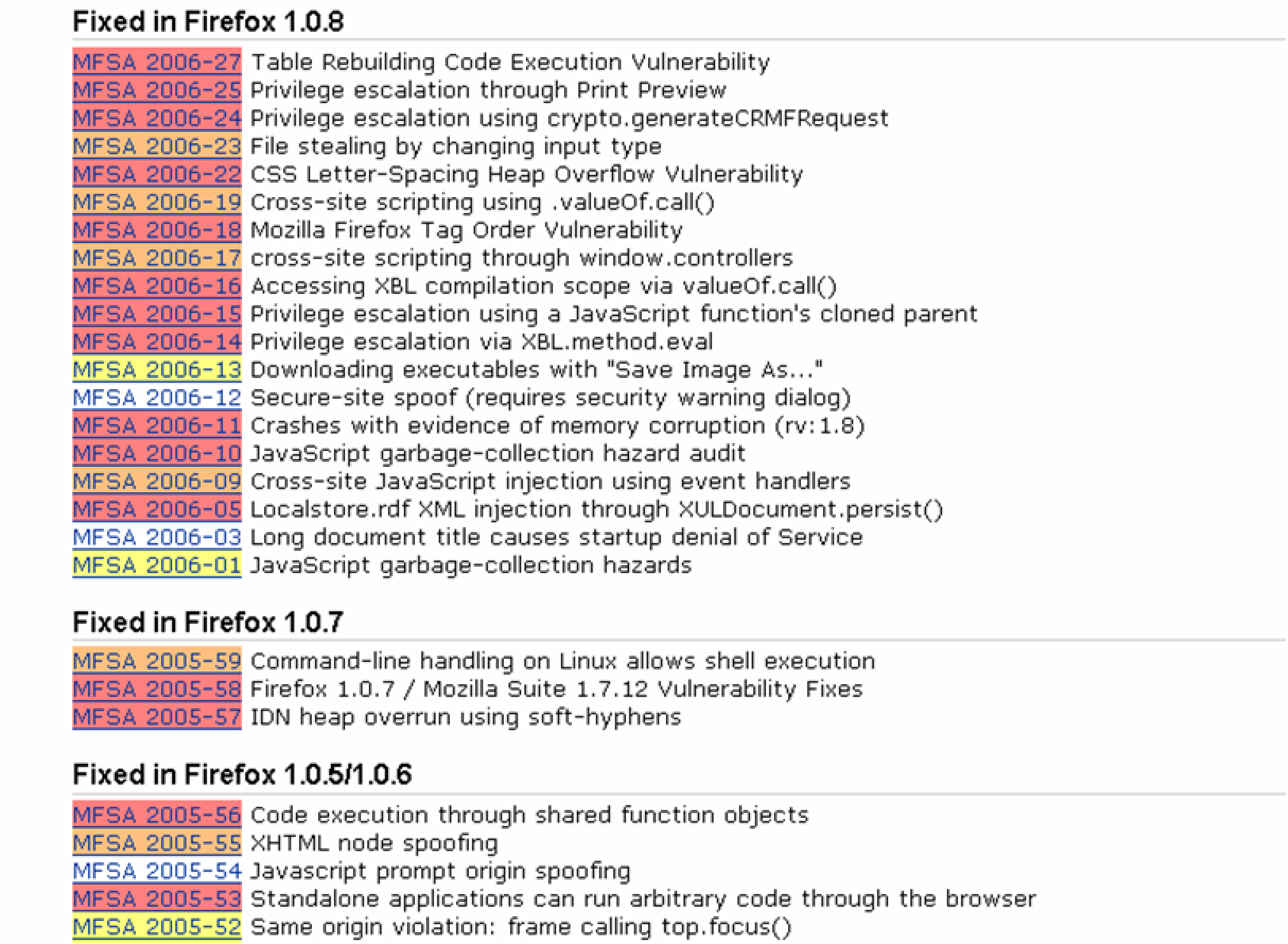
User Interface shows Petname and warns of ambiguities or key/nickname changes

The attacks of the future are against the brain. Software can help us detect semantic attacks – if it speaks out language, automates difficult tasks and offers us granular but workable control over authority. Diagram: zooko/miller

SECTION 6

Application Analysis

Vulnerability Database of Mozilla/Firefox: Recurring Bug Patterns



Fixed in Firefox 1.0.8

MFSA 2006-27 Table Rebuilding Code Execution Vulnerability
MFSA 2006-25 Privilege escalation through Print Preview
MFSA 2006-24 Privilege escalation using crypto.generateCRMFRequest
MFSA 2006-23 File stealing by changing input type
MFSA 2006-22 CSS Letter-Spacing Heap Overflow Vulnerability
MFSA 2006-19 Cross-site scripting using .valueOf.call()
MFSA 2006-18 Mozilla Firefox Tag Order Vulnerability
MFSA 2006-17 cross-site scripting through window.controllers
MFSA 2006-16 Accessing XBL compilation scope via valueOf.call()
MFSA 2006-15 Privilege escalation using a JavaScript function's cloned parent
MFSA 2006-14 Privilege escalation via XBL.method.eval
MFSA 2006-13 Downloading executables with "Save Image As..."
MFSA 2006-12 Secure-site spoof (requires security warning dialog)
MFSA 2006-11 Crashes with evidence of memory corruption (rv:1.8)
MFSA 2006-10 JavaScript garbage-collection hazard audit
MFSA 2006-09 Cross-site JavaScript injection using event handlers
MFSA 2006-05 Localstore.rdf XML injection through XULDocument.persist()
MFSA 2006-03 Long document title causes startup denial of Service
MFSA 2006-01 JavaScript garbage-collection hazards

Fixed in Firefox 1.0.7

MFSA 2005-59 Command-line handling on Linux allows shell execution
MFSA 2005-58 Firefox 1.0.7 / Mozilla Suite 1.7.12 Vulnerability Fixes
MFSA 2005-57 IDN heap overrun using soft-hyphens

Fixed in Firefox 1.0.5/1.0.6

MFSA 2005-56 Code execution through shared function objects
MFSA 2005-55 XHTML node spoofing
MFSA 2005-54 Javascript prompt origin spoofing
MFSA 2005-53 Standalone applications can run arbitrary code through the browser
MFSA 2005-52 Same origin violation: frame calling top.focus()

Evolution of Browser Security

Ur-Browser: Vulnerabilities

Browser with Hobbles: New Vulnerabilities

Browser with more Hobbles

Browser with Security Concepts: Same Origin

Browser with Security Mechanisms

Privilege Definitions/Policies/Sandbox

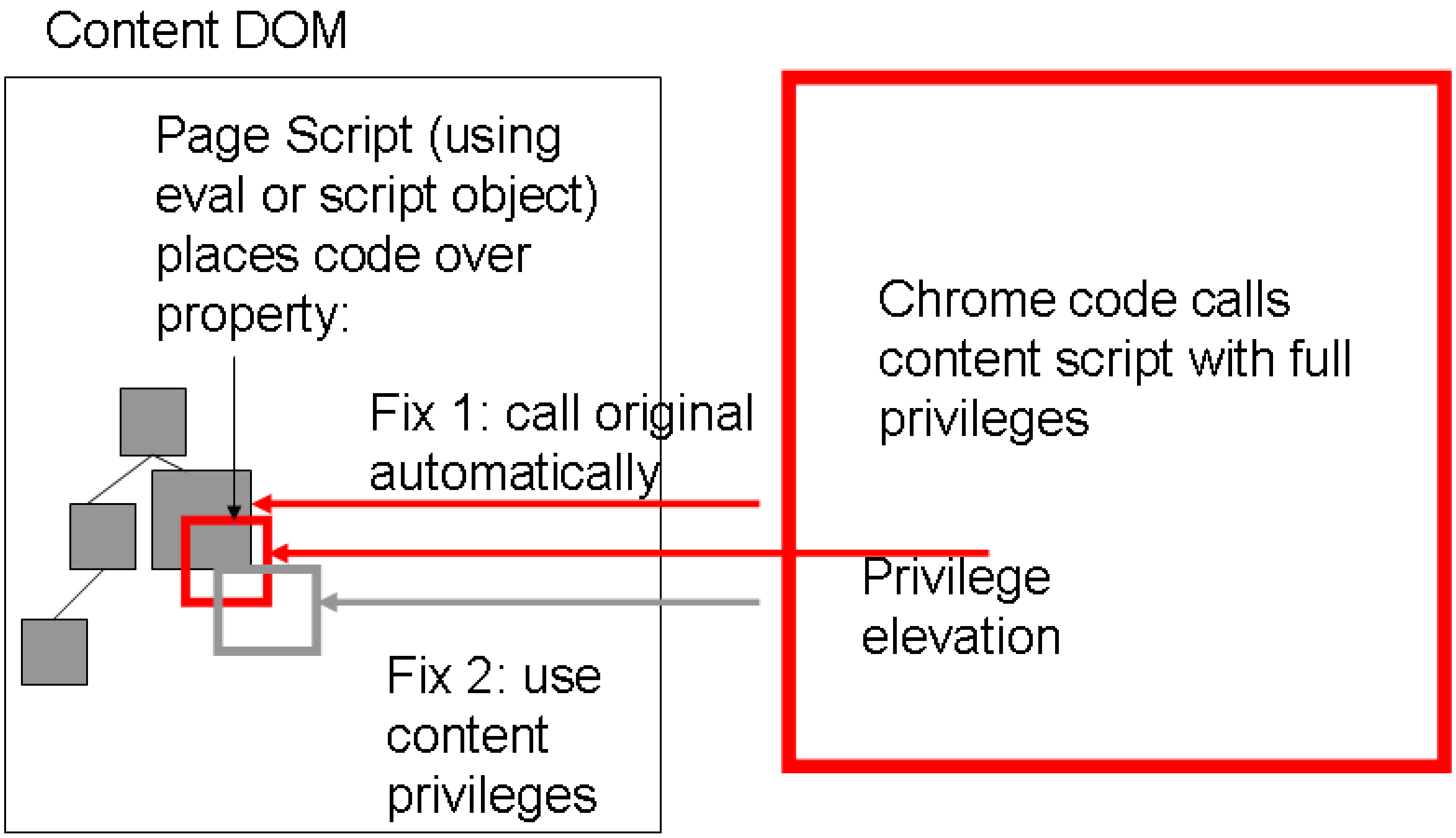
Data-Tainting

Code Signing (more Privileges for Trusted Code)

Modern Browser: New Vulnerabilities

POLA Browser: Authority Reduction

Interaction between privileged and non-privileged code



Content DOM

Page Script (using eval or script object) places code over property:

Fix 1: call original automatically

Fix 2: use content privileges

Chrome code calls content script with full privileges

Privilege elevation

SECTION 7

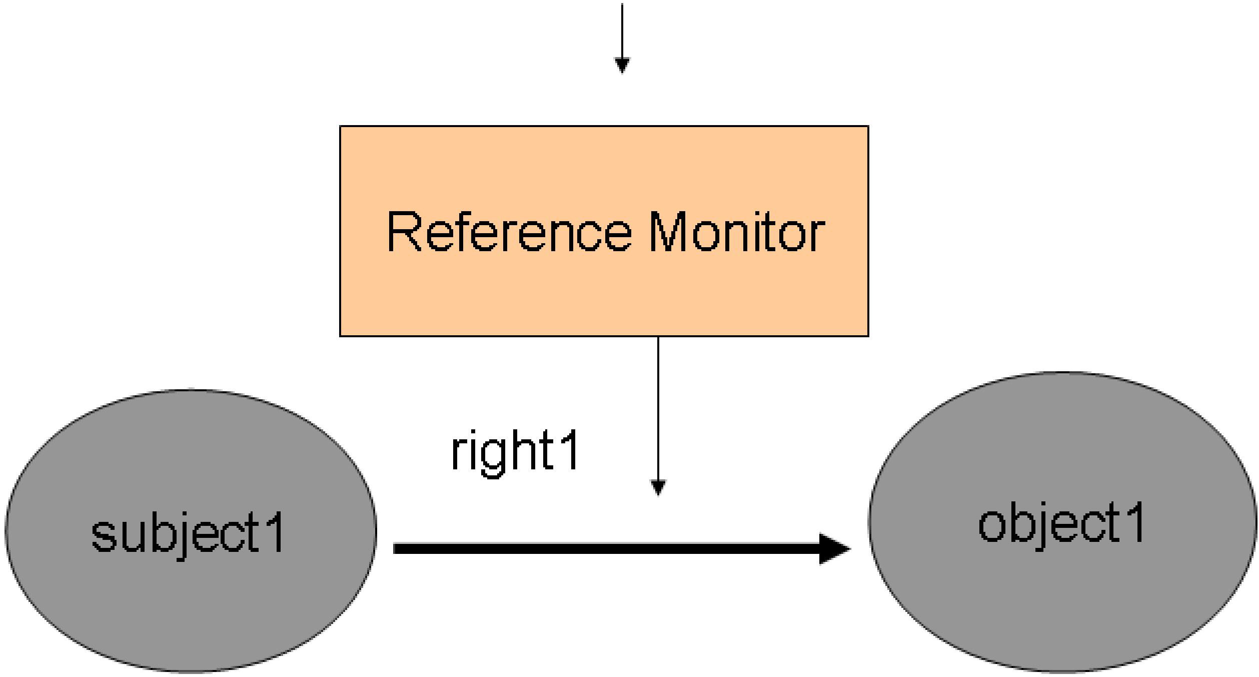
Modeling the Flow of Authority – Security Analysis

Access Control Matrix

Static Rules

	Object1	Object2
Subject1	Right1	Right2
Subject2	Right3	Right1

Software Configuration



The Access Control Matrix encodes access rights between subjects and objects.

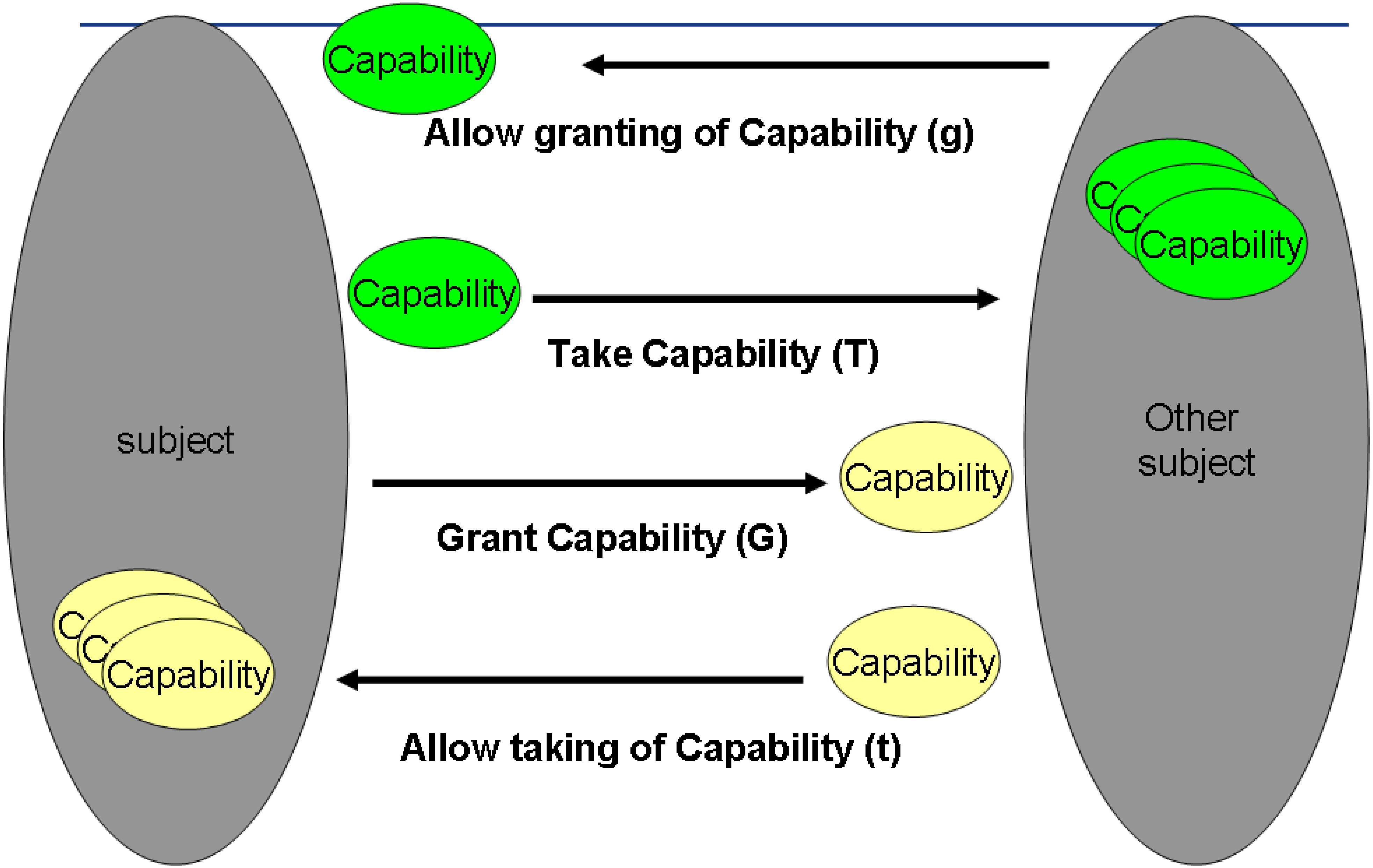
Primitives: create, delete [Subject|Object], enter, delete [Right]

```
createFile(subject, fileObject) {
        create FileObject
        enter Right=Own in ACM[subject,fileObject]
        // more rights…
}

transerferRead(OwnerSubject, OtherSubject,  fileObject) {
        // more conditions…
        if (ACM[subject,fileObject] == ReadRight)
                enter ReadRight in ACM[OtherSubject,fileObject]
}
```

Manipulations of the ACM cannot be verified computationally (halting problem)

Take/Grant Model for the Flow of Authority



Allow granting of Capability (g)

Take Capability (T)

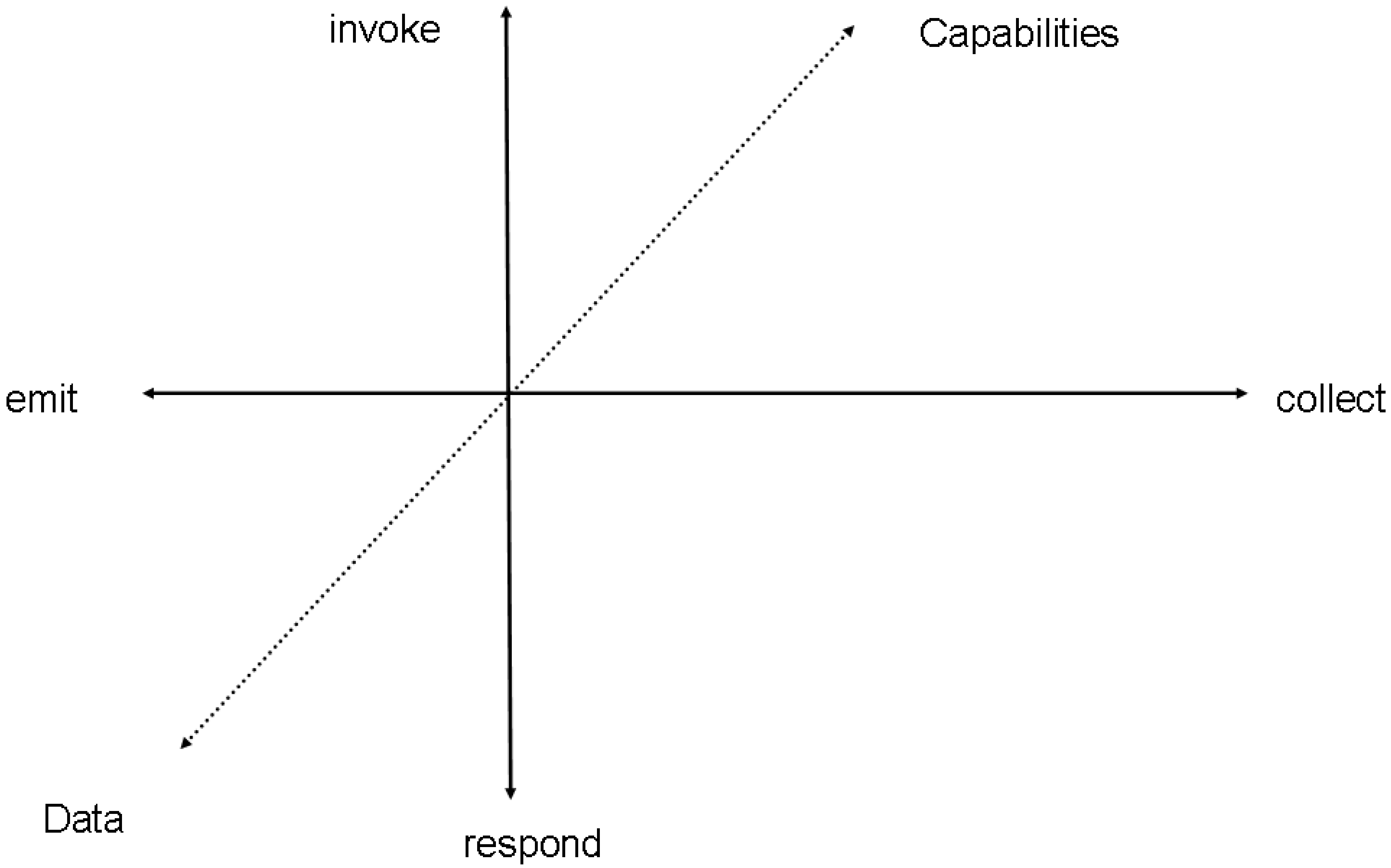
Grant Capability (G)

Allow taking of Capability (t)

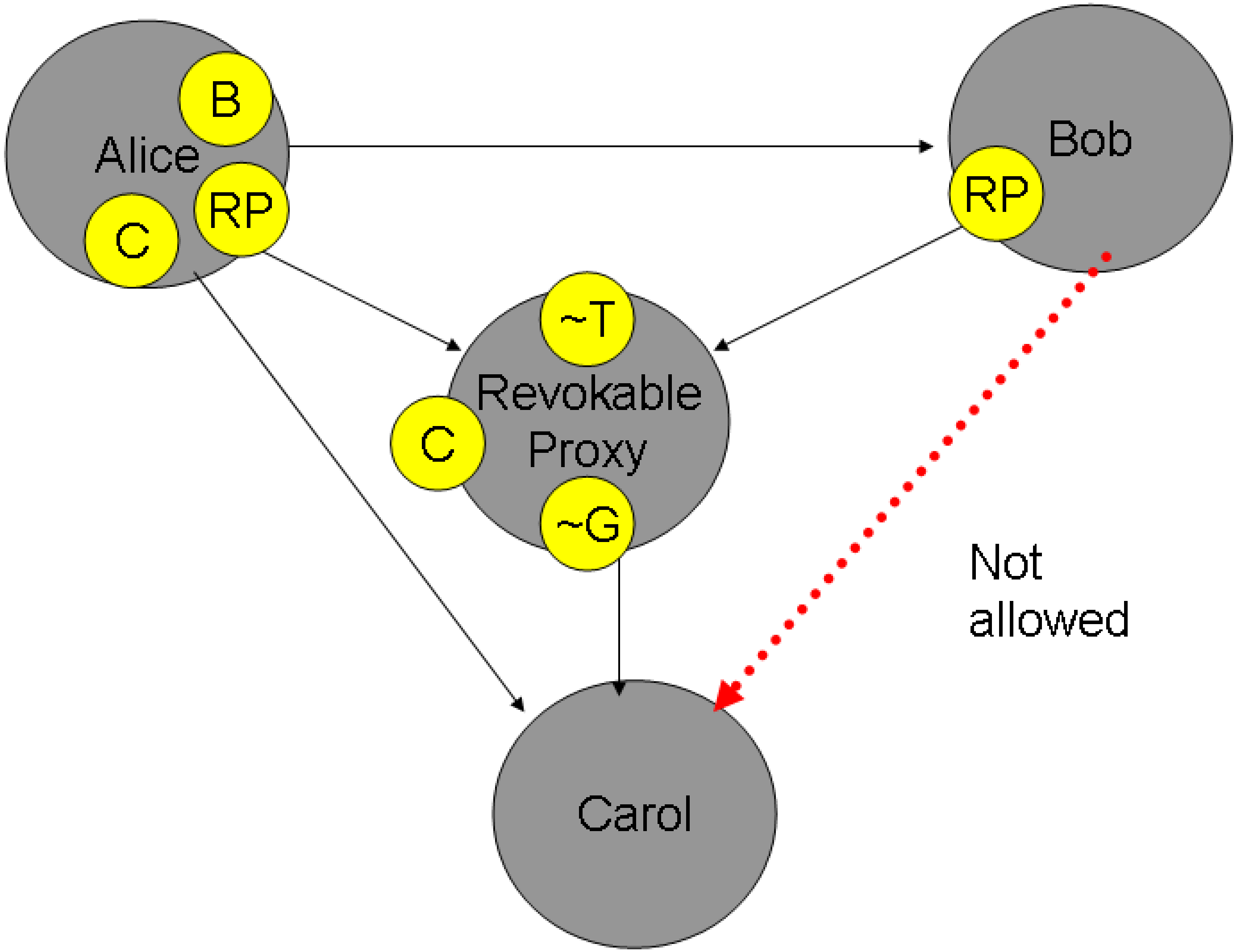
Possible subject behavior: T, G, t, g

(Continued)

Dimensions of extended Take/Grant Behavior Models

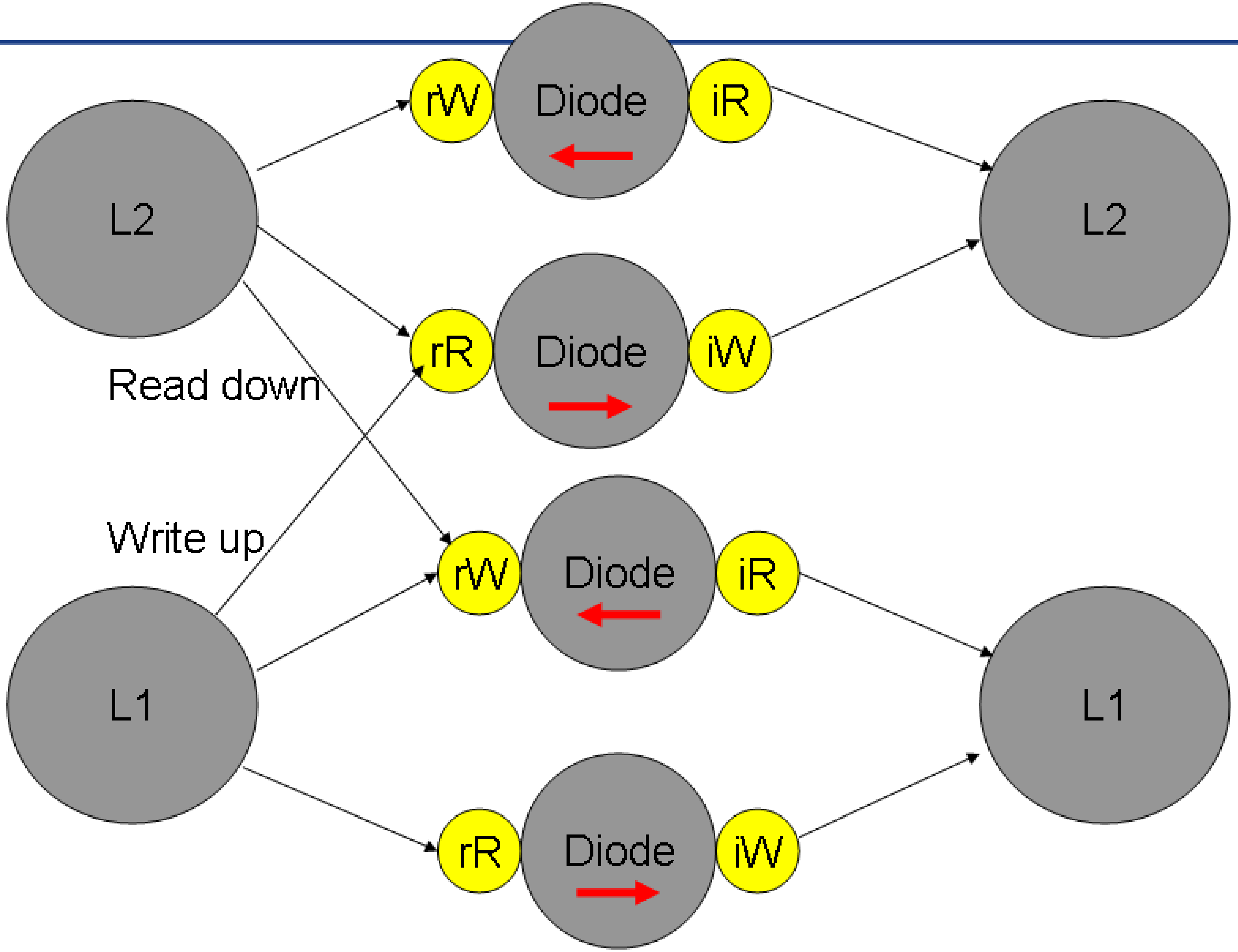


Caretaker Model with extended Take/Grant semantics



RP: Do not expose Carol, only pass capabilities acquired through collaboration

Simulation of *-properties with ext. Take/Grant

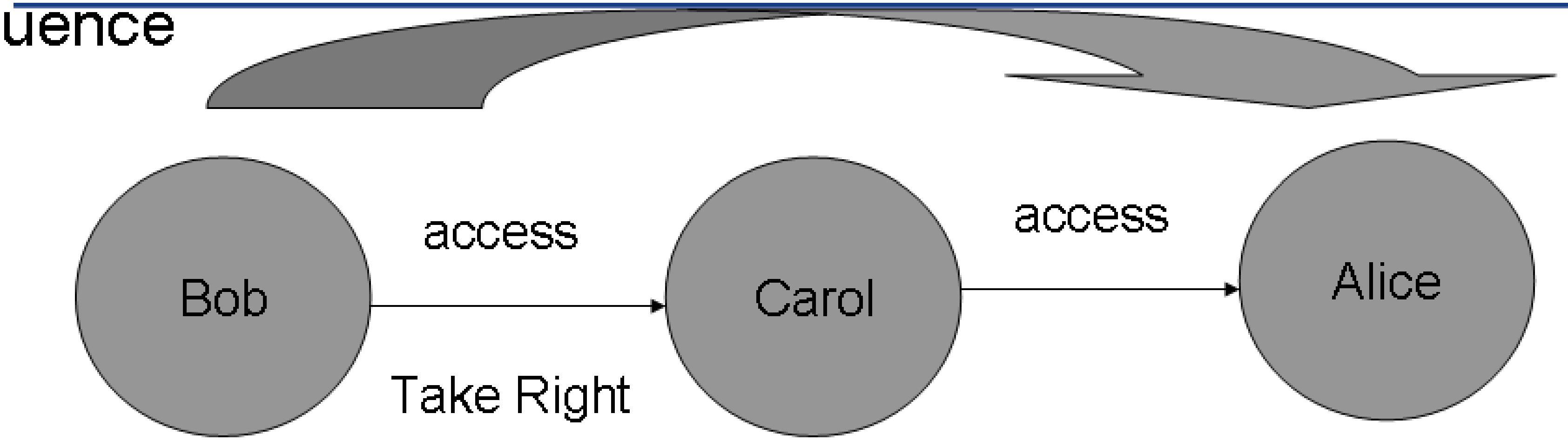


Read-Down Diode: rW/iR (L1< L2); Write-Up Diode: rR/iW (L2 > L1)

Only data, no authority (capability), no assumptions about L1/L2 behavior

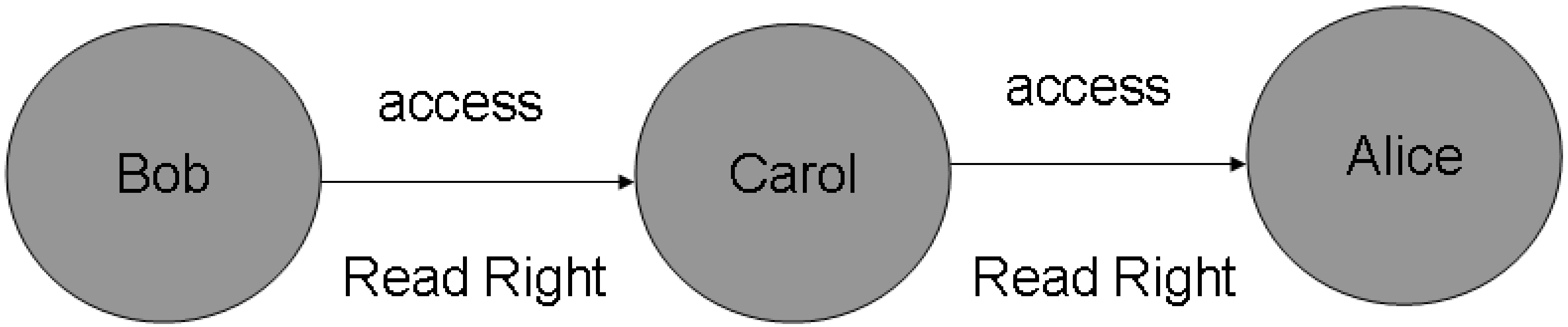
De-Jure Influence

Authority Propagation (Bob gets access to Alice and can propagate that access

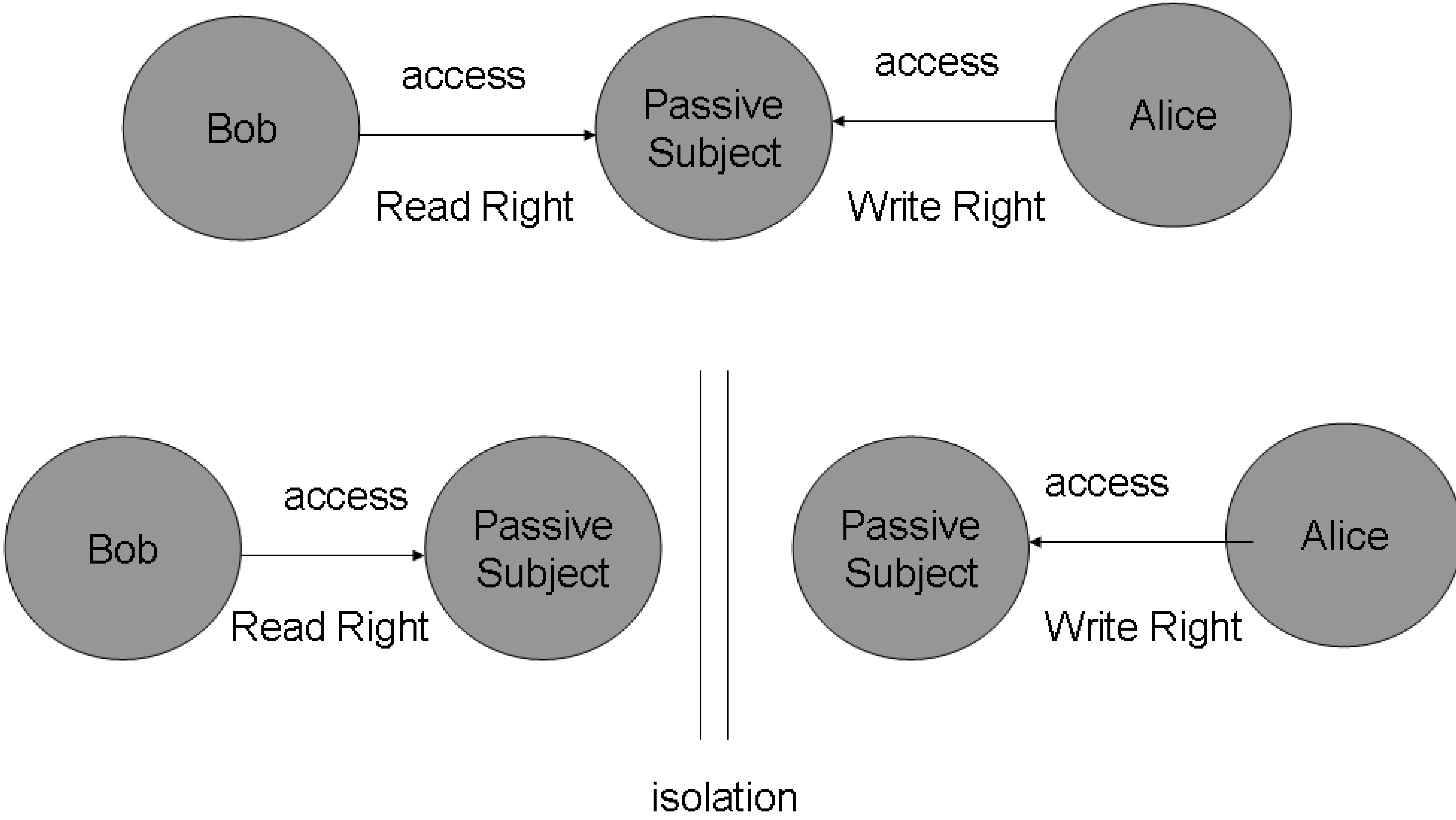


De-Fakto Influence

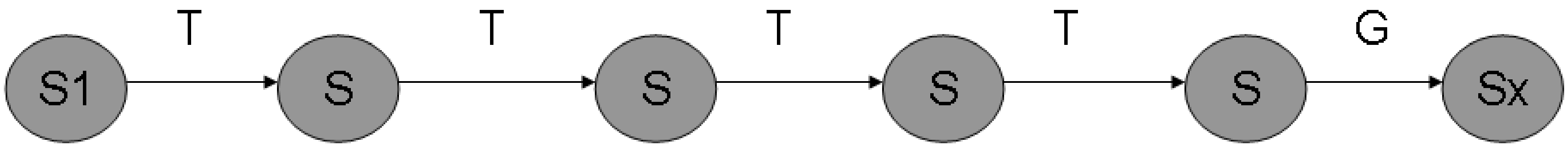
Propagation of influence (Data from Alice can influence Bob but Bob does not get authority over Alice)



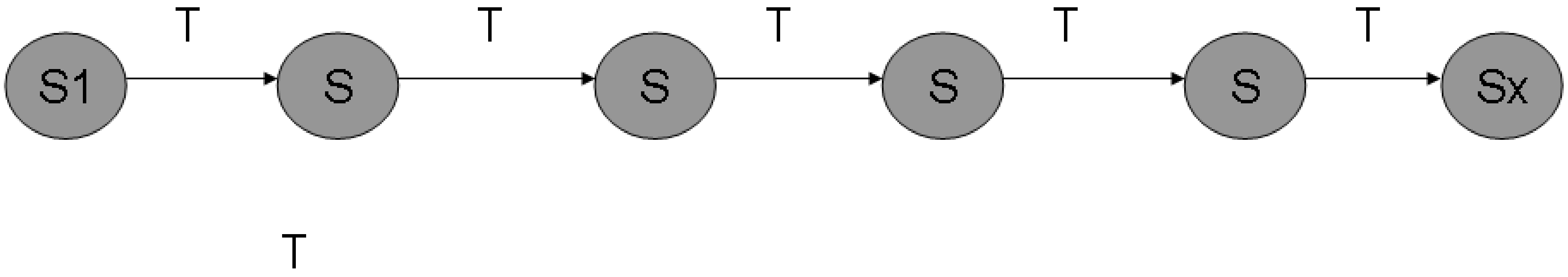
Isolation with two passive objects



Can S1 give Sx some right?



Can S1 get some right from Sx?



SECTION 8

Security and Usability – Chances for a better GUI on a reduced authority system.

Intentions and System Behavior

User concepts vs.
System concepts

Damage minimization
strategies in software

Separation of concerns:
users vs. system

Manipulation of many
objects

Currently user expectations and system behavior do not match. How do we get around useless warning dialogs?

Resources

1. Virtualization: Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". *Communications of the ACM* 17 (7): 412 –421.
2. Secure languages and Systems: www.erights.org
3. Alternative Multithreading Approaches: CSP, Event loops, Stefan Reich, Escape from Multi-threaded Hell http://www.drjava.de/e-presentation/html-english/img0.html. Peter Welch, A CSP Model for Java Threads www.cs.kent.ac.uk/projects/ofa/jcsp/csp-java-model.pdf
4. Testing, Fuzzing: Month of the browser bug, Daniel Bachfeld, Die Axt im Walde http://www.heise.de/security/artikel/print/76512
5. Platform Security: Andy Tanenbaum et.al, can we make operating systems reliable and secure? www.computer.org
6. Robust Composition: Mark Miller Thesis, 2006 http://www.erights.org/talks/thesis/index.html
7. Gates Talk RSA 2005 (www.rsa.com)
8. Darpa Browser Architecture (www.combex.com)
9. Authority Reduction, Theoretical Foundations and Decidability: www.combex.com (powerbox Concept, secure desktop etc.
10. Concept Based Education: Peter van Roy, Saif Haridi, Concepts, Technologies and Models of Computer Programs
11. Usability and Security, Simson Garfinkel Thesis 2005, Cranor and Garfinkel 2006
12. Petname Systems: http://zooko.com, Mark Stiegler, An Introduction to Petname Systems, http://www.skyhunter.com/marcs/petnames/IntroPetNames.html
13. Safety Analysis: Fred Spiessens, Peter Van Roy, A Practical Formal Model for Safety Analysis in Capability Based Systems
14. Functional Safety Standard: IEC 61508, International Electrotechnical Commission Functional safety of electrical/electronic/programmable electronic safety-related systems.

    http://www.iee.org/oncomms/pn/functionalsafety/HLD.pdf