Linux Security Modules

SELinux, AppArmor & Tomoyo trough security models

-

Kernel Recipes 2013

Previously on KR Season 1

Previously on KR Season 1

- Formal models for computer security
- Specify functional & assurance requirements → CC
- Implementation
- Testing → CC

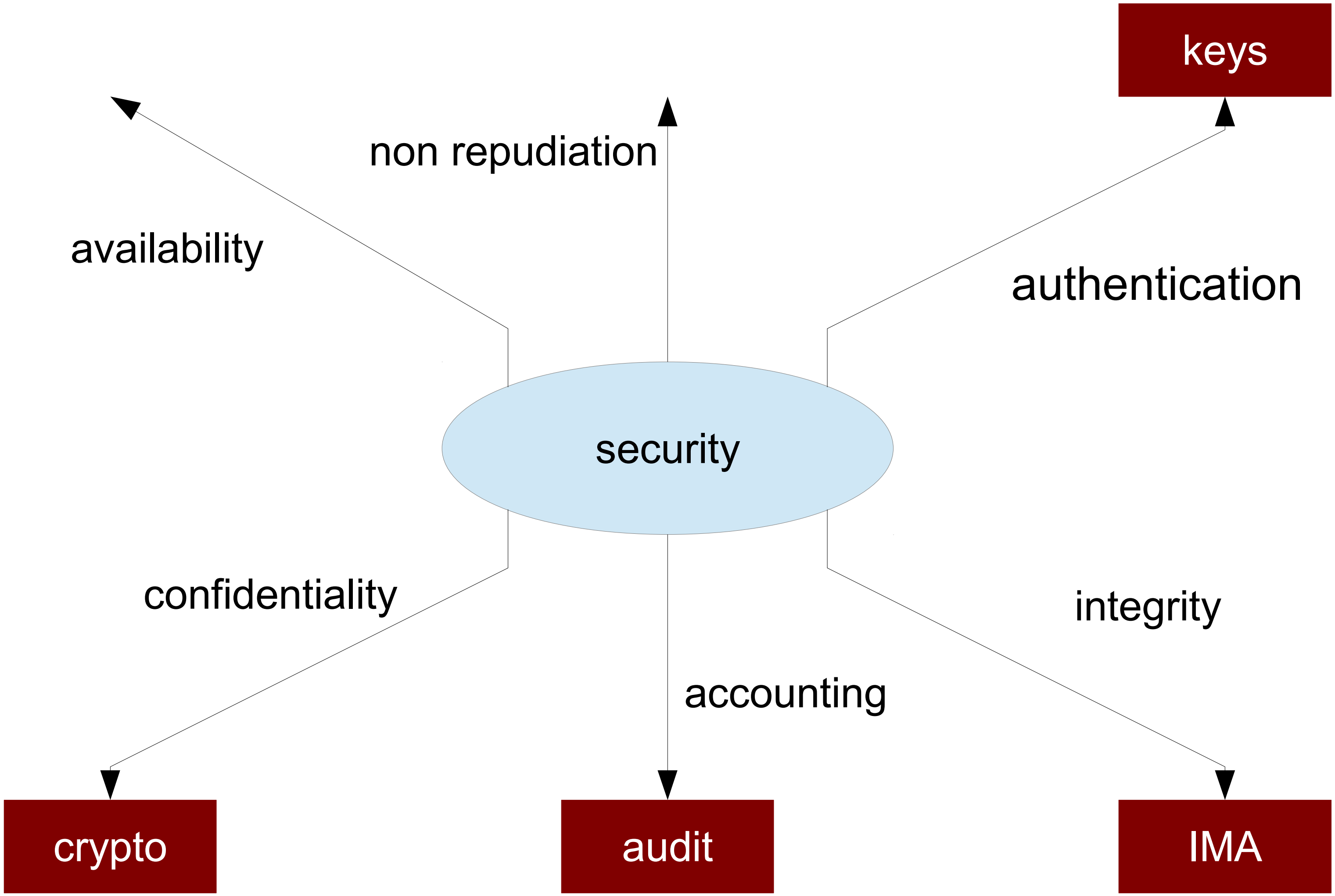
CC = Common Criteria

Previously on KR Season 1

- LOMAC : Low Water-Mark Mandatory Access Control - 2000
- Bell-La Padula (BLP) – 1973
- object-capability - 1981
- Take-grant - 1977
- Biba – 1977
- Access control Matrix – 1971
- ..

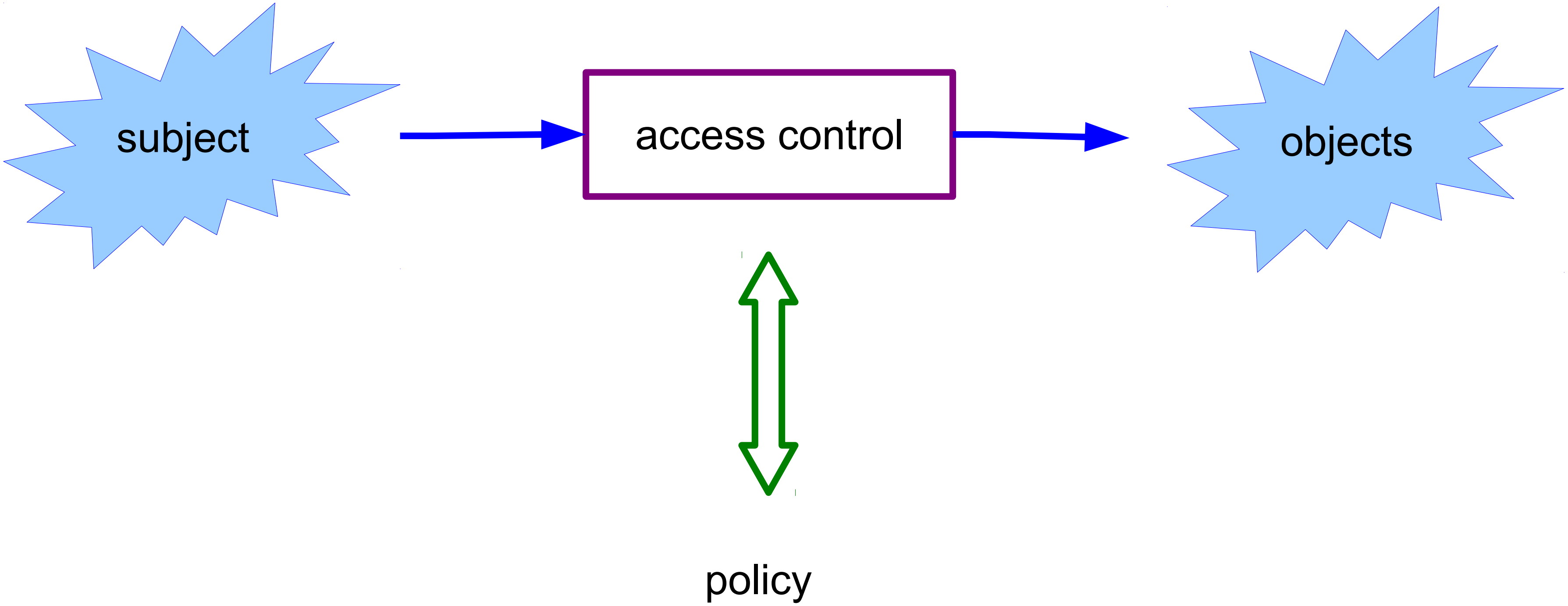
Previously on KR Season 1

Shared properties



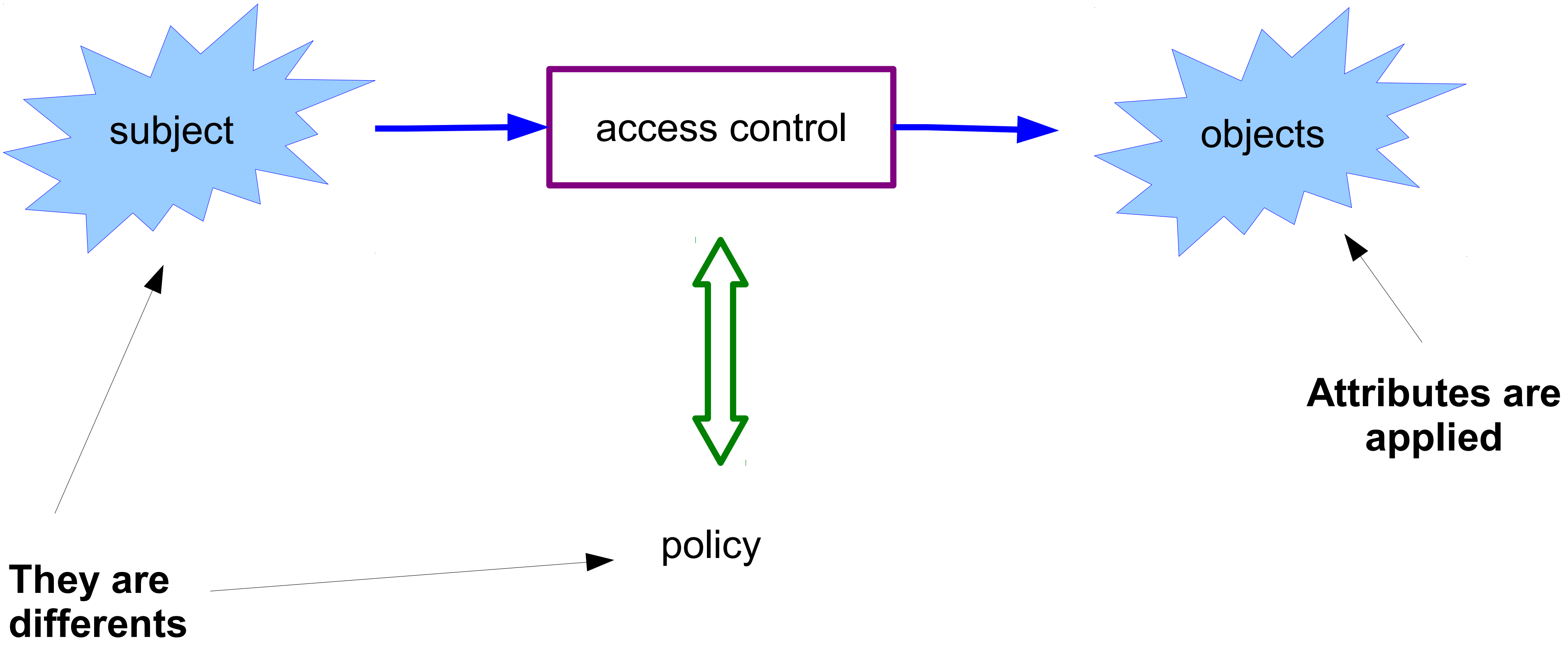
Previously on KR Season 1

What is MAC ?



Previously on KR Season 1

What is MAC ?



subject

access control

objects

policy

Attributes are applied

They are differents

Summary

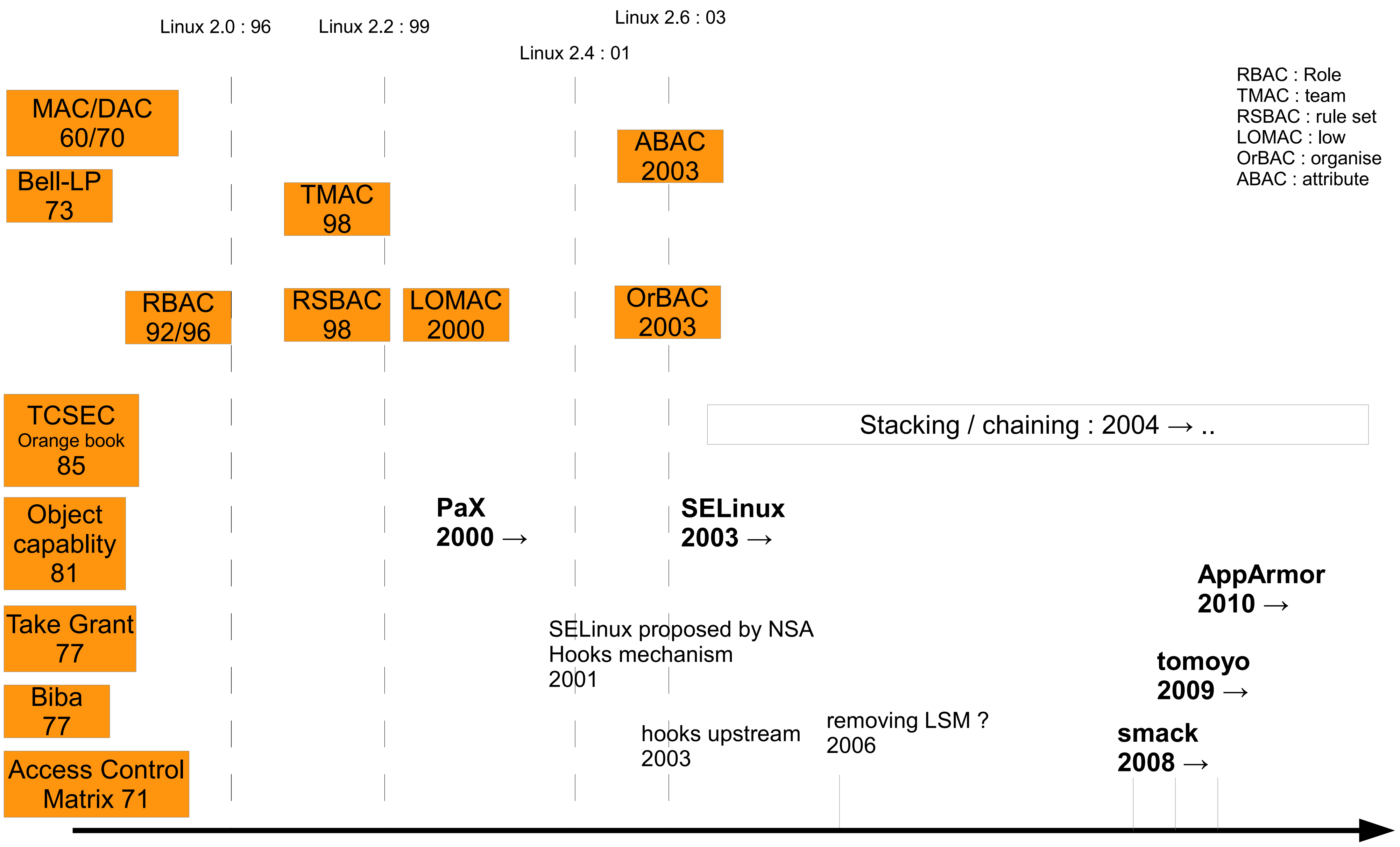
Summary

- Model for SELinux
  - History & discuss
- Model for AppArmor
  - History & discuss
- Model for Tomoyo
  - History & discuss

- Summary of the Linux Security Summit 2013 meeting

- Discuss about using LSM hooks for “information flow”

Access Control: timetable



Linux 2.0 : 96

Linux 2.2 : 99

Linux 2.4 : 01

Linux 2.6 : 03

MAC/DAC 60/70

Bell-LP 73

RBAC 92/96

TCSEC Orange book 85

Object capablity 81

Take Grant 77

Biba 77

Access Control Matrix 71

TMAC 98

RSBAC 98

LOMAC 2000

ABAC 2003

OrBAC 2003

Stacking / chaining : 2004 → ..

PaX 2000 →

SELinux 2003 →

SELinux proposed by NSA
Hooks mechanism
2001

hooks upstream 2003

removing LSM ? 2006

AppArmor 2010 →

tomoyo 2009 →

smack 2008 →

RBAC : Role
TMAC : team
RSBAC : rule set
LOMAC : low
OrBAC : organise
ABAC : attribute

SELinux

Model for SELinux : history

- NSA was the original developer
- Implementation of the operating system security architecture called Flask
- In the 2.5.x series, LSM framework was developed,so SELinux was ported for 2.6.0

- Flask : Flux Advanced Security Kernel

SELinux model : the Flask architecture

- Flask architecture simply implements MAC
- Principle of *“least privilege”*
- Objects and subjects are related to security attributes inside a “*security context”*
- Dealing with security context is not easy, so we can refer to it with a SID : security identifier, a kind or pointer, reference to the context.

  Exemple : it's working well for persistent objects
- A security decision can be made with {SID(subject), SID(object)}.
- Two kind of decisions exist :
  - Labeling decision : obj/sub transition → creating new file from directory
  - Access decision : check permissions for operations using Access Vector Cache (AVC) : access vector gives decisions for all permissions for a object, or directly on the server policy

SELinux model : the Flask architecture

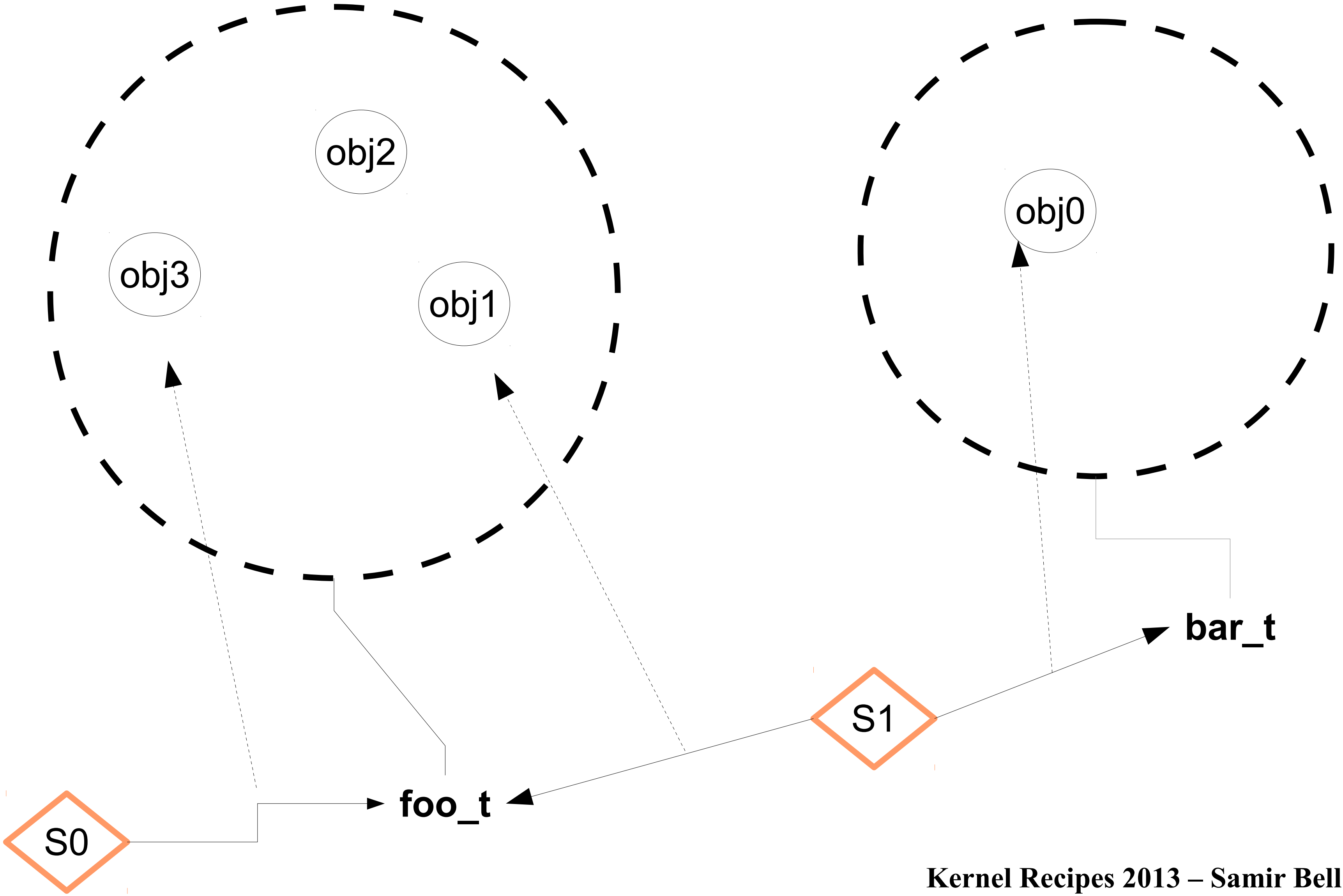
- Security policy over process and objects
- True innovation : splitting the technical architecture from the policy (not only a modularity)
- Demonstration by implementing :
  - Type enforcement (TE) 1980-1985
  - Role Based Access Control (RBAC) 1992-1996
  - Multi Level Security (MLS)

SELinux model : TE – type enforcement

- SAT : Secure Ada Target, 1st implementation, late 1980s
- Labels (security informations) on subjects and objects
- security context with labels on subjects → “domain label” (DTE)
- security context with labels on objects → “type label” (DTE)
- class exist for using objects directly:
  - Same type, but different class → can manage the situation
- TE uses *role* for users, not domain.
  - credentials mechanism → b6dff3 : separate task security context from task_struct, so no more true label on subject
- TE enables the labeling decisions and the access decisions

SELinux model : TE – type enforcement

- obj3, obj1 and obj2 are in the same type “foo_t”



SELinux model : type enforcement

- “So it's all about classification ?”
  - I think so, but it is not really a shared idea..

SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles

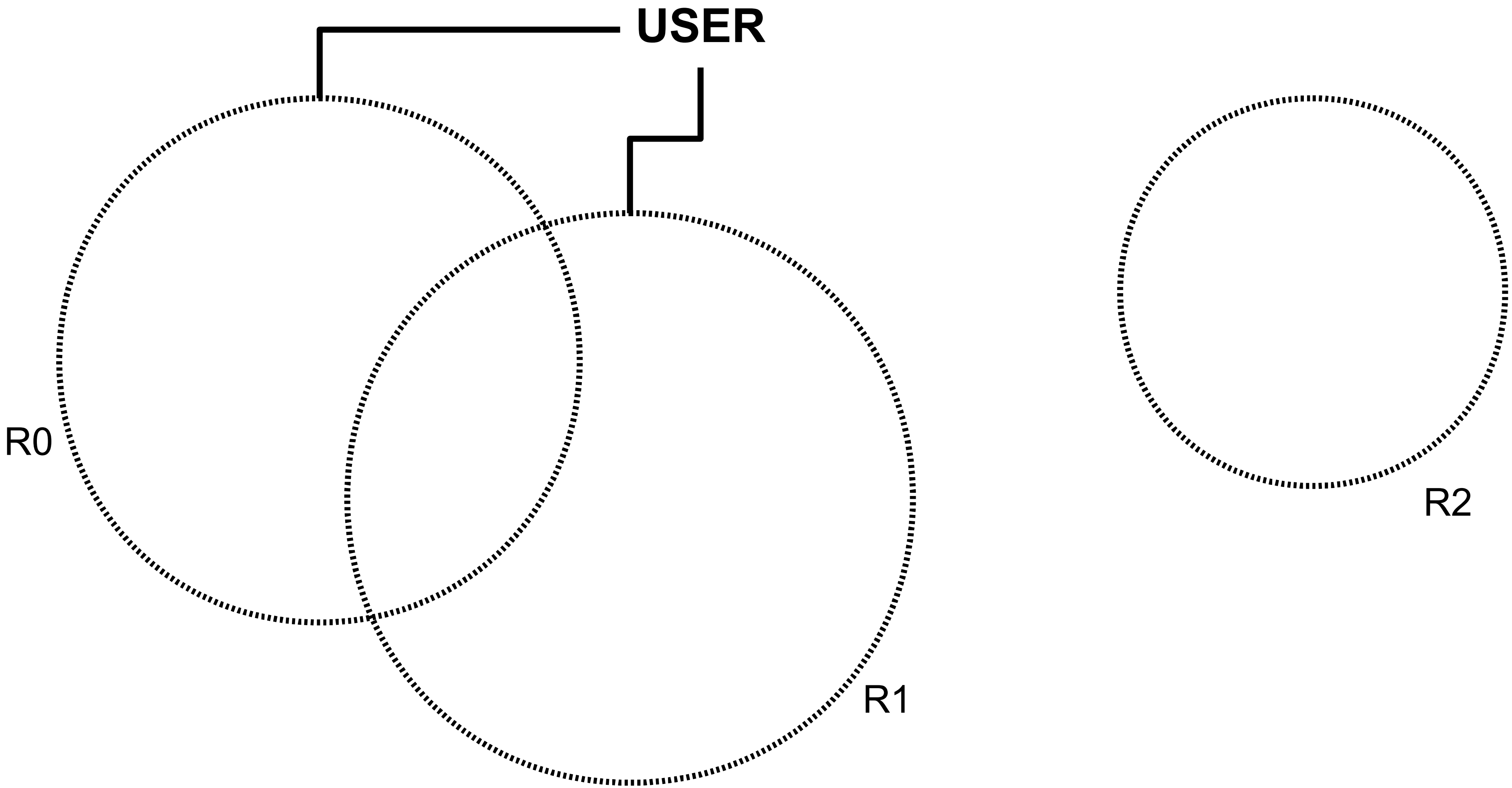
SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles

USER

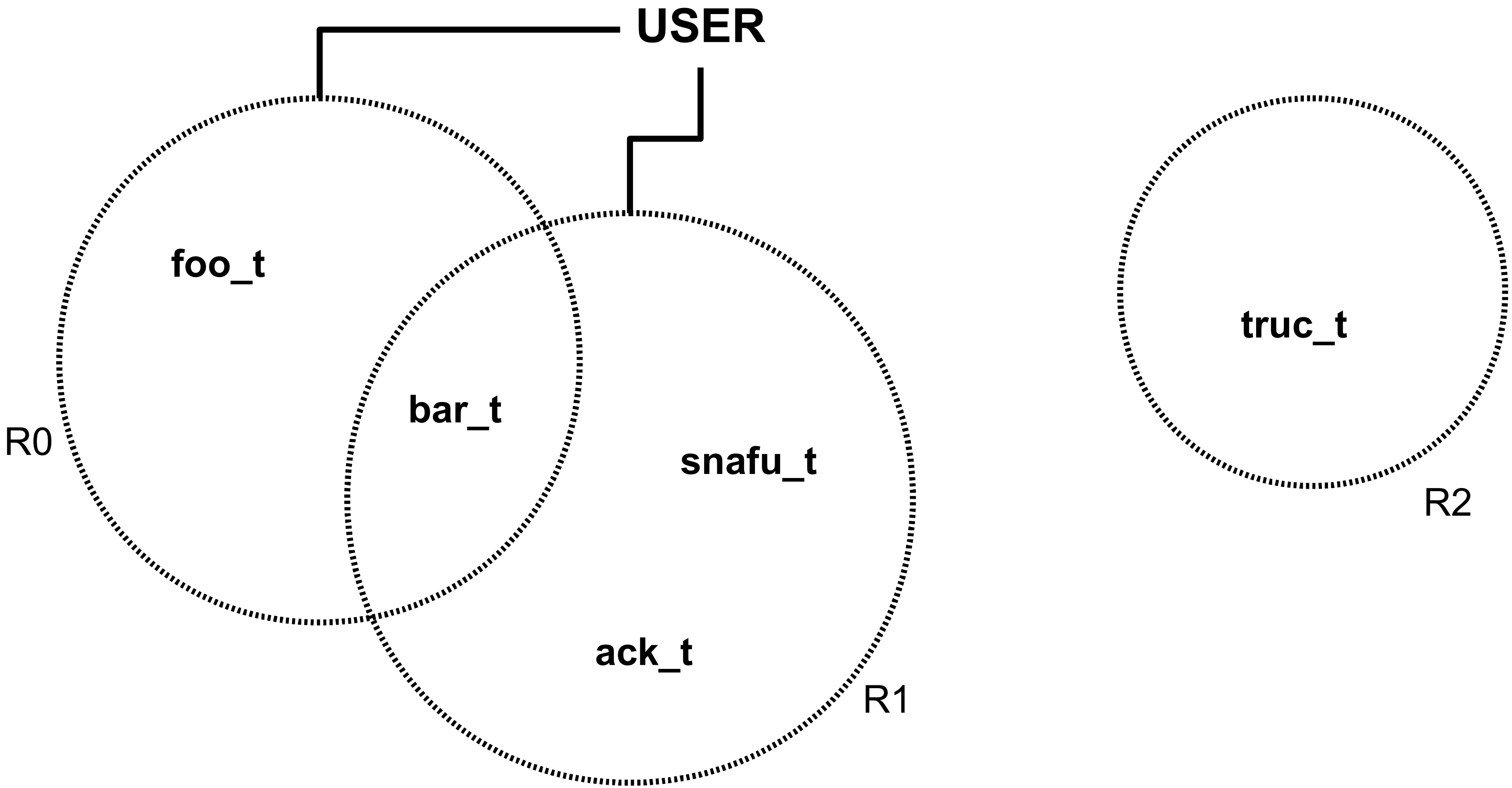
SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles



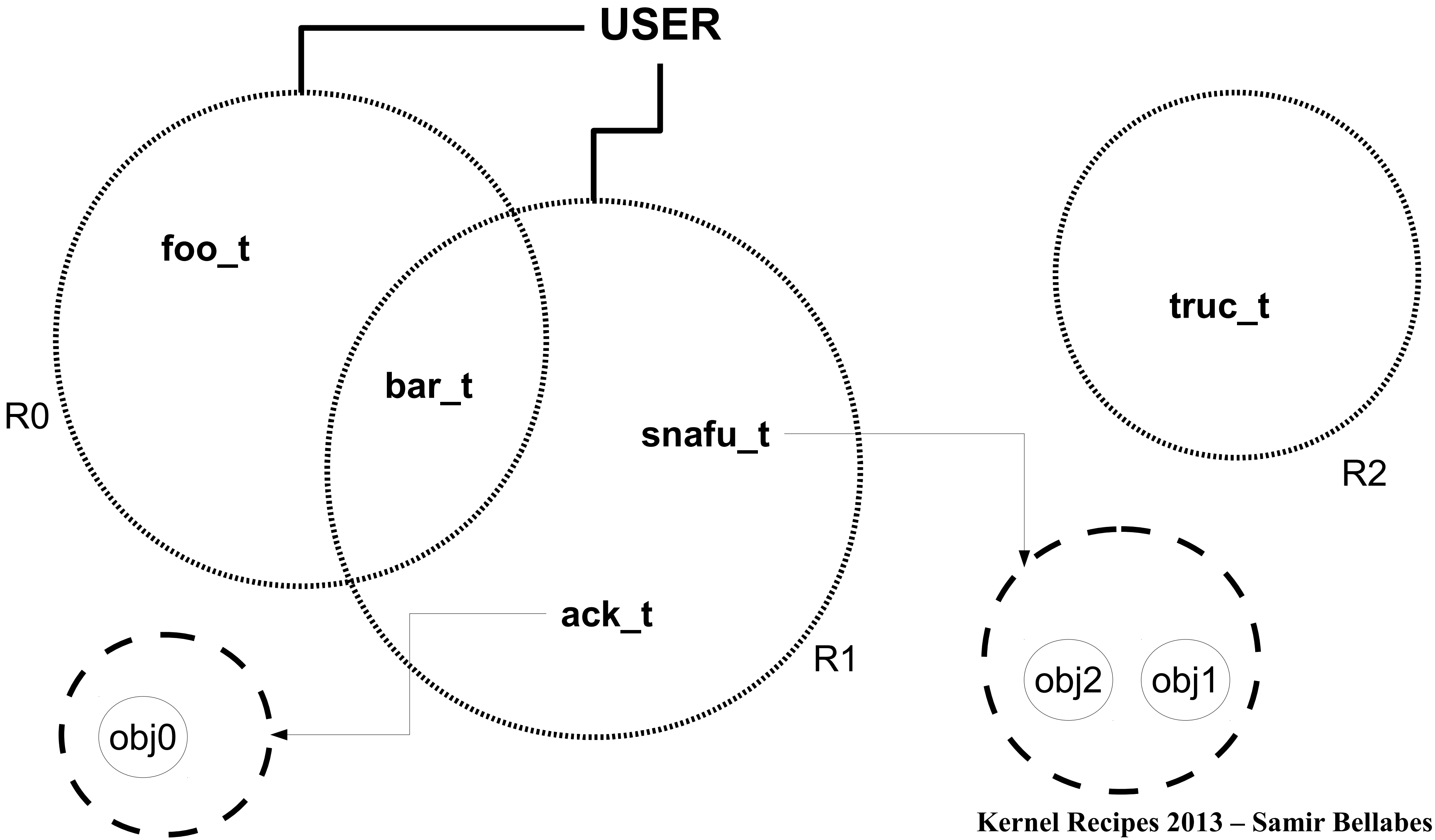
SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles



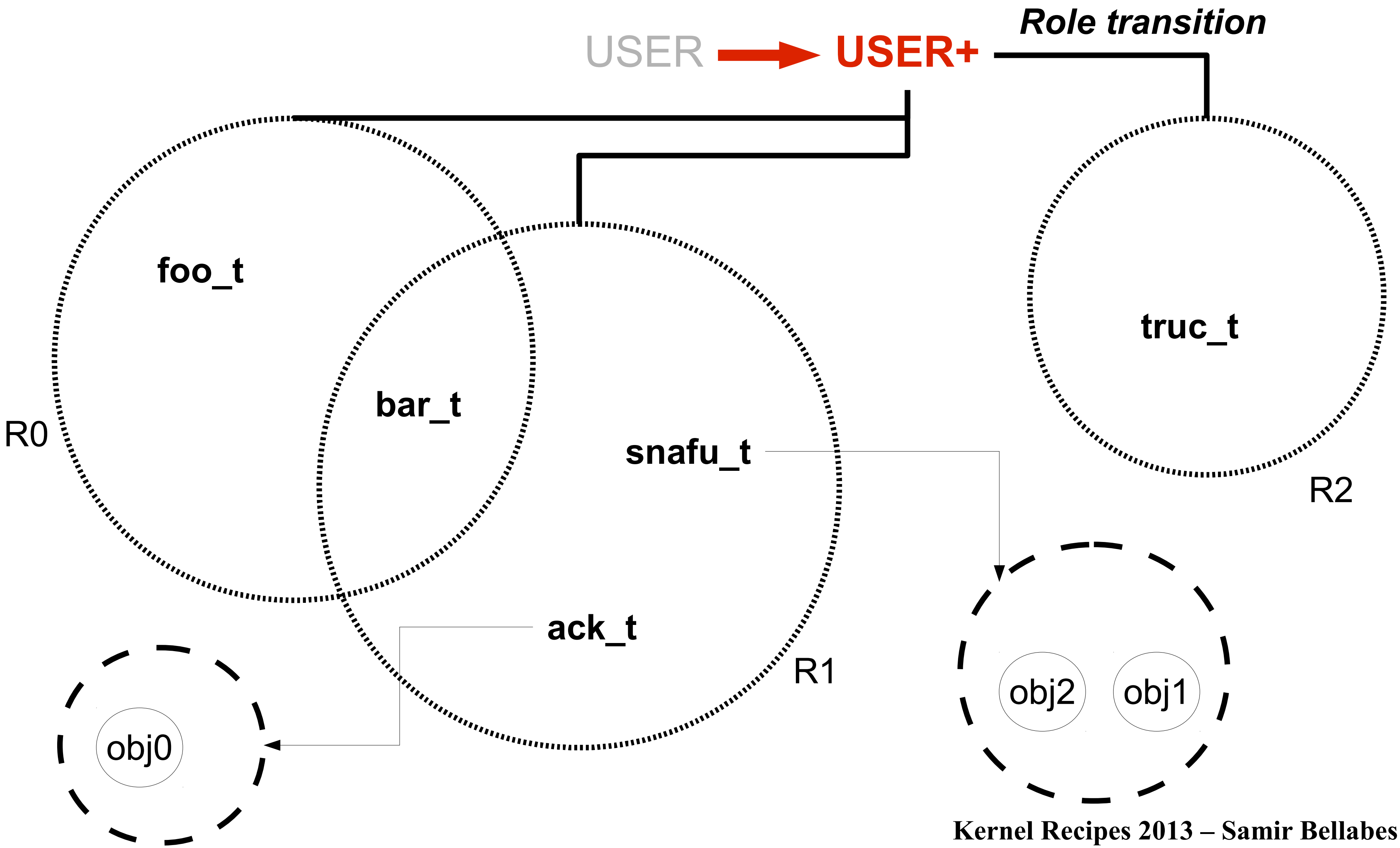
SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles



SELinux model : RBAC

- RBAC : Role Based Access Control
- Attaching *roles* on users, attaching *permissions* on roles

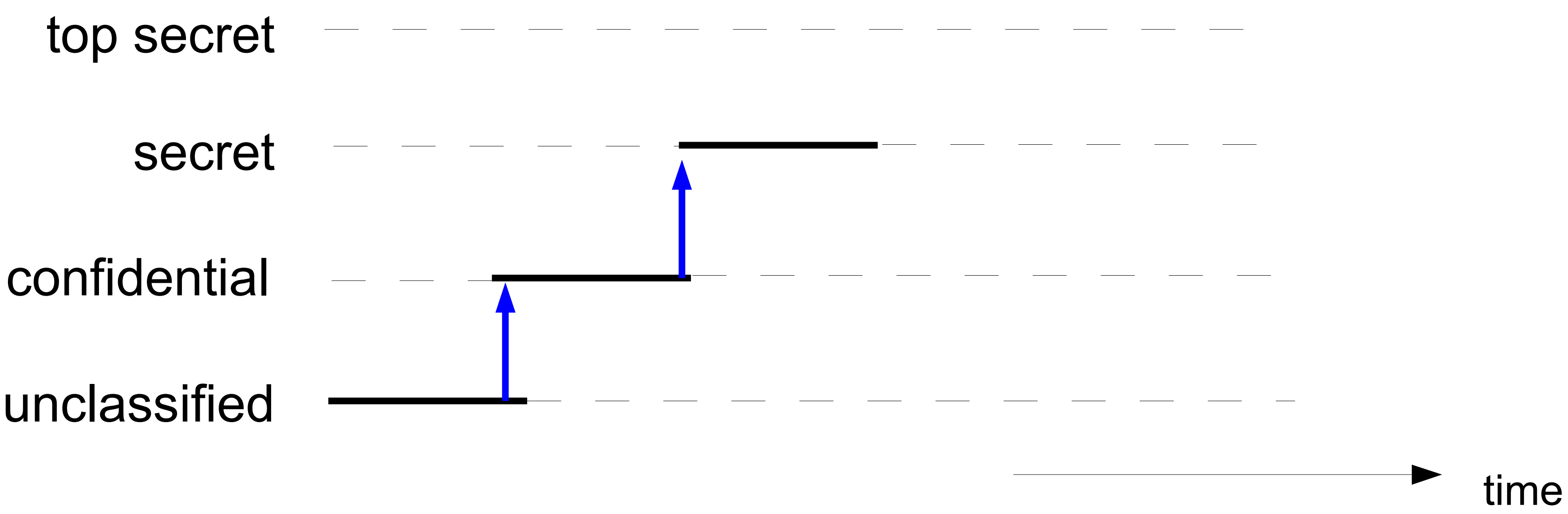


SELinux model : MLS

- It's about security levels
- SELinux implements Bell-Lapadula model

SELinux model : MLS

- It's about security levels
- SELinux implements Bell-Lapadula model

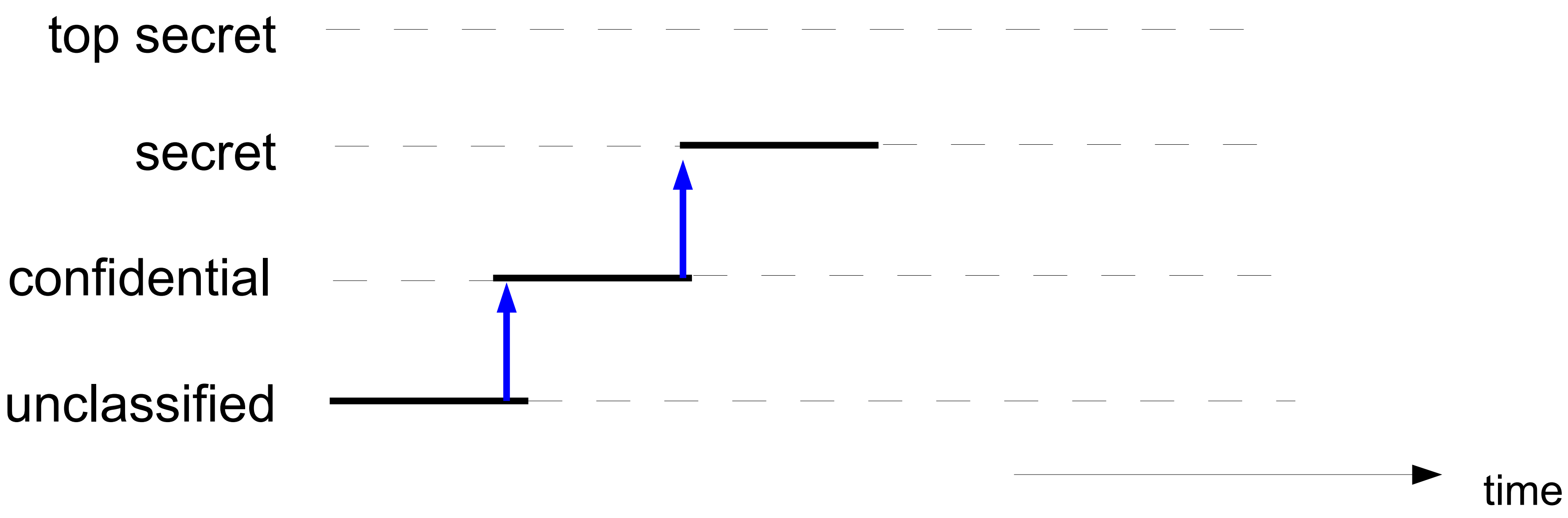


Transition states are managed

SELinux model : MLS

- It's about security levels
- SELinux implements Bell-Lapadula model

Transition states are managed



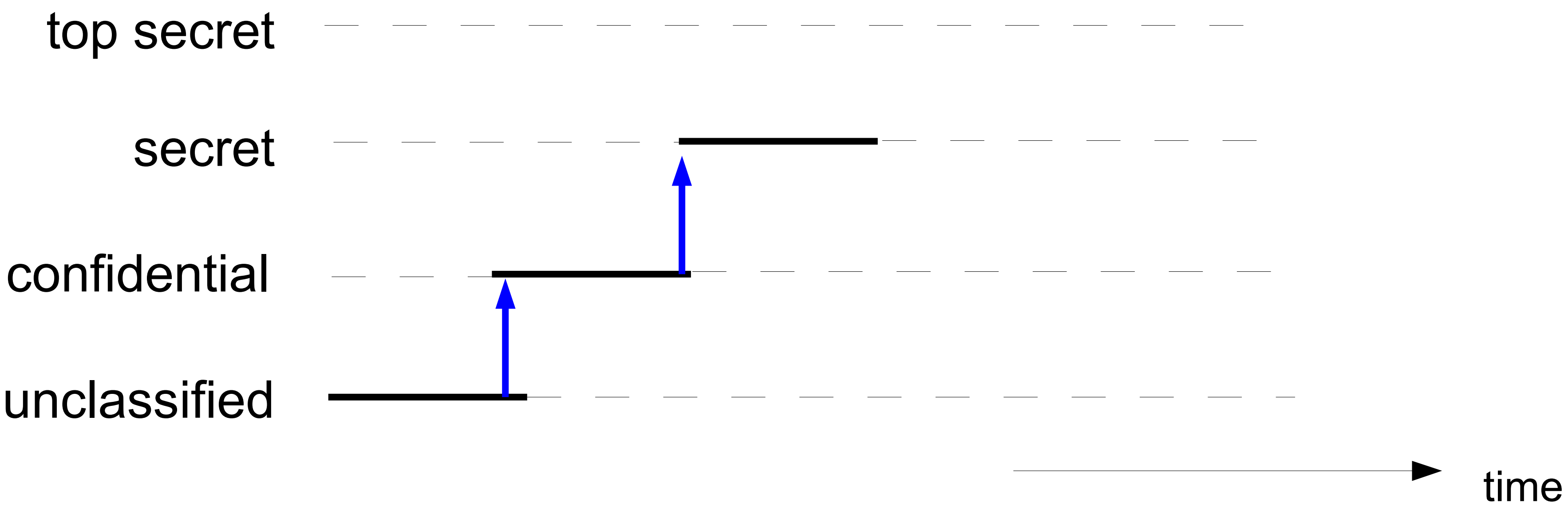
Read-down : Security(subject) > Security(object)

write-up: Security(subject) < Security(object)

SELinux model : MLS

- It's about security levels
- SELinux implements Bell-Lapadula model

Transition states are managed



Read-down : Security(subject) > Security(object)

*Opposite is **Biba** for integrity*

write-up: Security(subject) < Security(object)

SELinux : booting

- Booting / quit is a real deal : assure reliability on security is hard (embedded, ...).

- start_kernel()
- security_init()
- Initial SID (1)
- Initialize AVC, selinuxfs
- Set enforcing mode from config
- (some stuff called relabeling)
- Start /sbin/init with label context

AppArmor

Model for AppArmor : history

- Originally from 1998
- Upstream in 2.6.36

AppArmor model : type enforcement

- A modified domain type enforcement (again) : **Profile** is the domain type
  - Normally subject ↔ objects ↔ permissions (type enforcement)
  - But profile A = { (obj0, perm0), (obj1, perm1), .. }
  - Profiles are stored in database

- Using information labels on objects (void *security) until creds patches (2.6.29)
- For files, AppArmor is using path-name as information, no label (dealing with mount point) (called *implicit labeling)*
- Using a technical mean called *“deriving implicit types”* ..

Tomoyo

Tomoyo model : type enforcement

- Process are attached a single domain
- If a process exec a program, divide or transit the domain
- Operations granularity on objects are “read/write/execute”

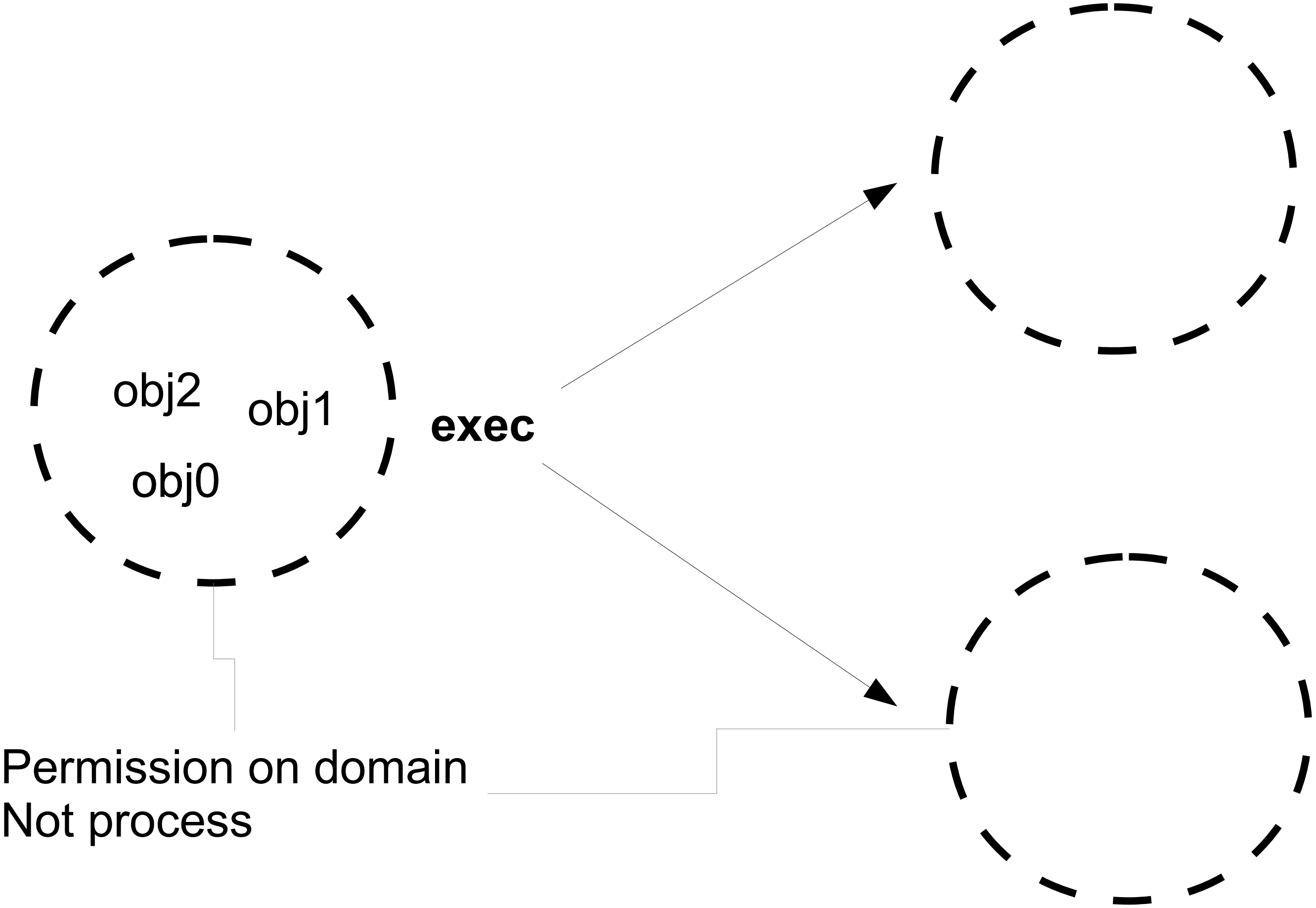
Tomoyo model : domain → path-named

- Starting with domain <*kernel*>
- Domain for /sbin/init is <*kernel>/sbin/init/*
- Exemple :
  - *<kernel>/sbin/init/etc/rc.d/service*
  - *<kernel>/usr/sbin/sshd/bin/bash*

- There are some exceptions (restarting services no more <*kernel*>/..)

Tomoyo model : type enforcement

- Process are attached a single domain
- If a process exec a program, divide or transit the domain
- Operations granularity on objects are “read/write/execute”



Permission on domain
Not process

Model for Tomoyo : history

- As far as I remember : Fighting
- Revive “void *security” : b6dff3
- Hook for network : post_accept
- Merging
- ..

Summary of Linux Security Summit 2013

Summary of LSS 2013

- Update on all security modules.
- Security mechanisms : ASLR, anti-patterns : using PaX plugins for gcc (!), using Coccinelle (!!!!),
- Stacking (agaaaaain..) but now it's called *multiple concurrent security models*
- technical papers for embedded
- http://kernsec.org/wiki/index.php/Linux_Security_Summit_2013

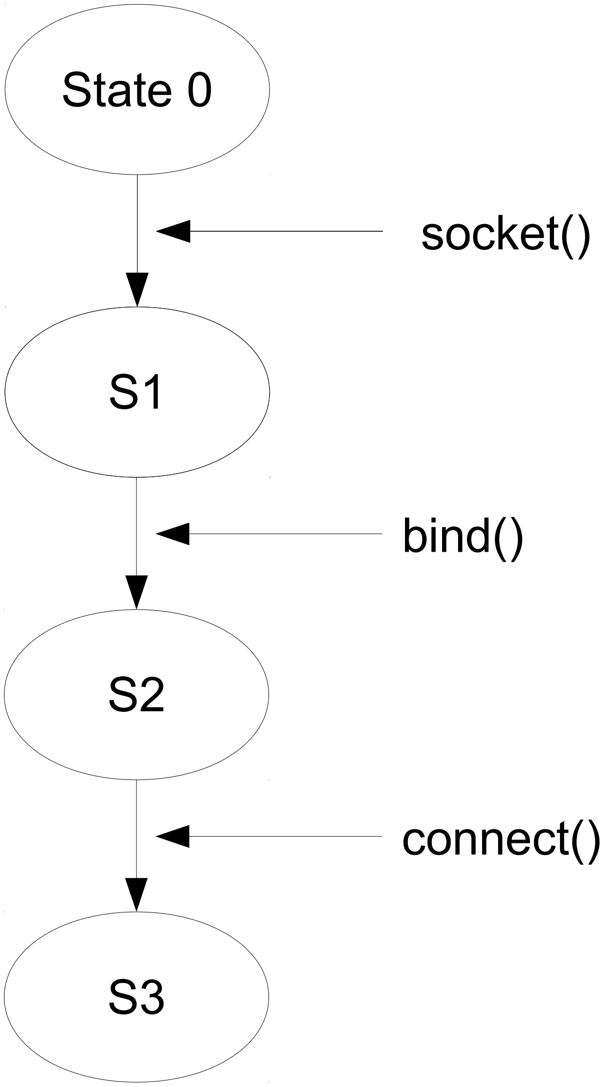
Using LSM hooks for “information flow”

Using LSM hooks for “information flow”

Entering #no_bullshit zone
Thanks Gandi for sponsoring Kernel Recipes

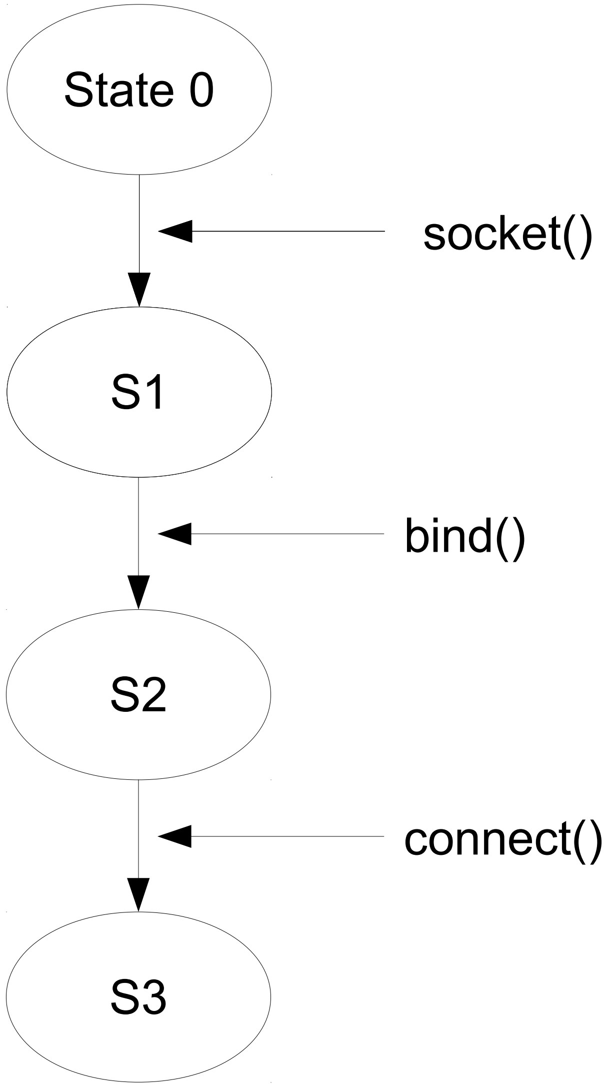
Information flow with hooks ?

- It's all about state machine and transitions



Information flow with hooks ?

- It's all about state machine and transitions



How can we build this interesting kind of graphs ?

Why not using LSM hooks as “borders” ?

Information flow with hooks ?

because ghosts are among us !

Information flow with hooks ?

- Let's take a memory buffer
- There are lots of functions which can modify m
  - write(m,..), mmap(m,..), str*(m,..)
- Let's say you **can** actually don't miss a function which can modify m and you can put a trap (hook) inside all this functions.
- So now you can have the graph ..

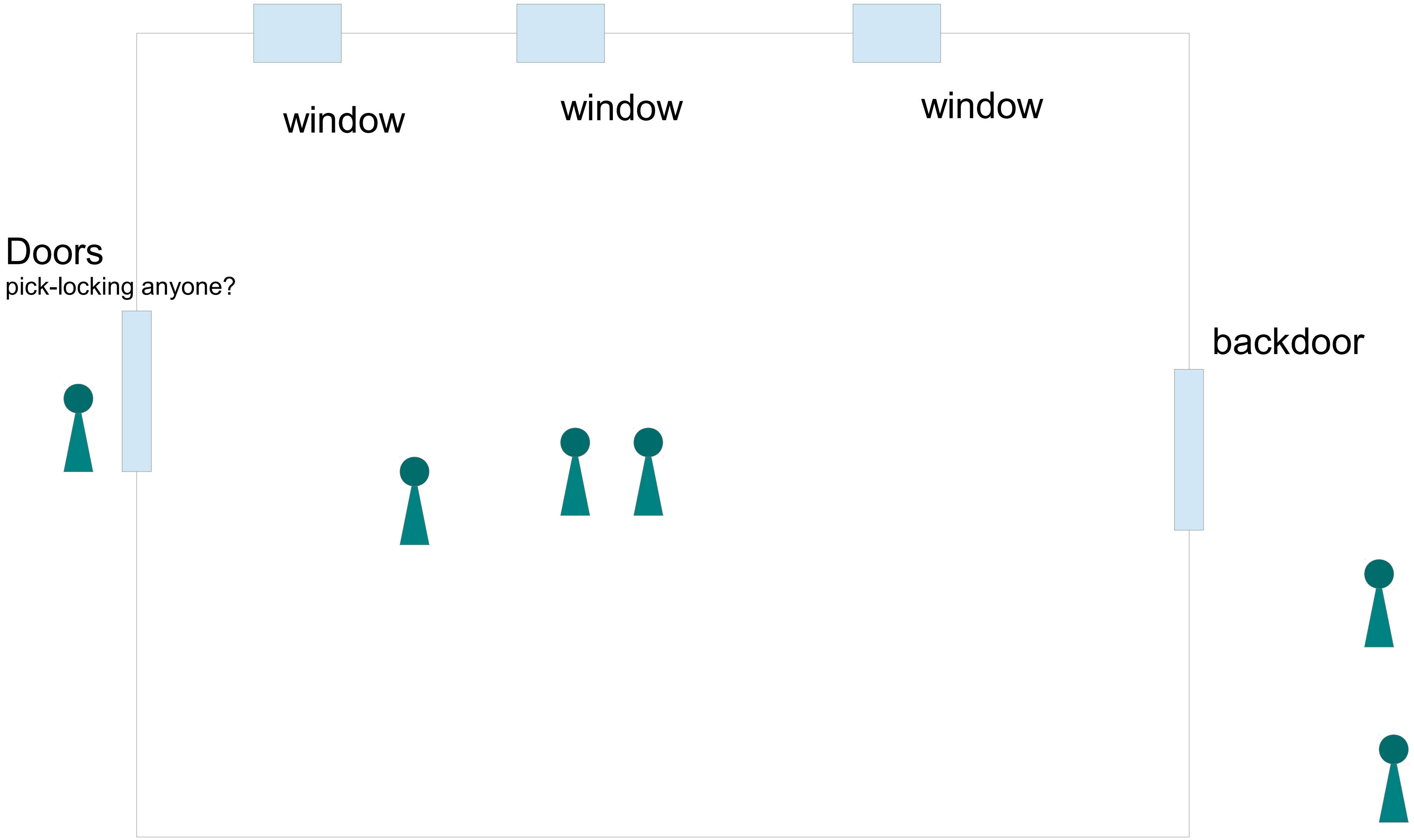
Information flow with hooks ?

- Let's take a memory buffer
- There are lots of functions which can modify m
  - write(m,..), mmap(m,..), str*(m,..)
- Let's say you **can** actually don't miss a function which can modify m and you can put a trap (hook) inside all this functions.
- So now you can have the graph ..

- What about ***m[10] = 0;*** ??
- How can you hook this operation ?

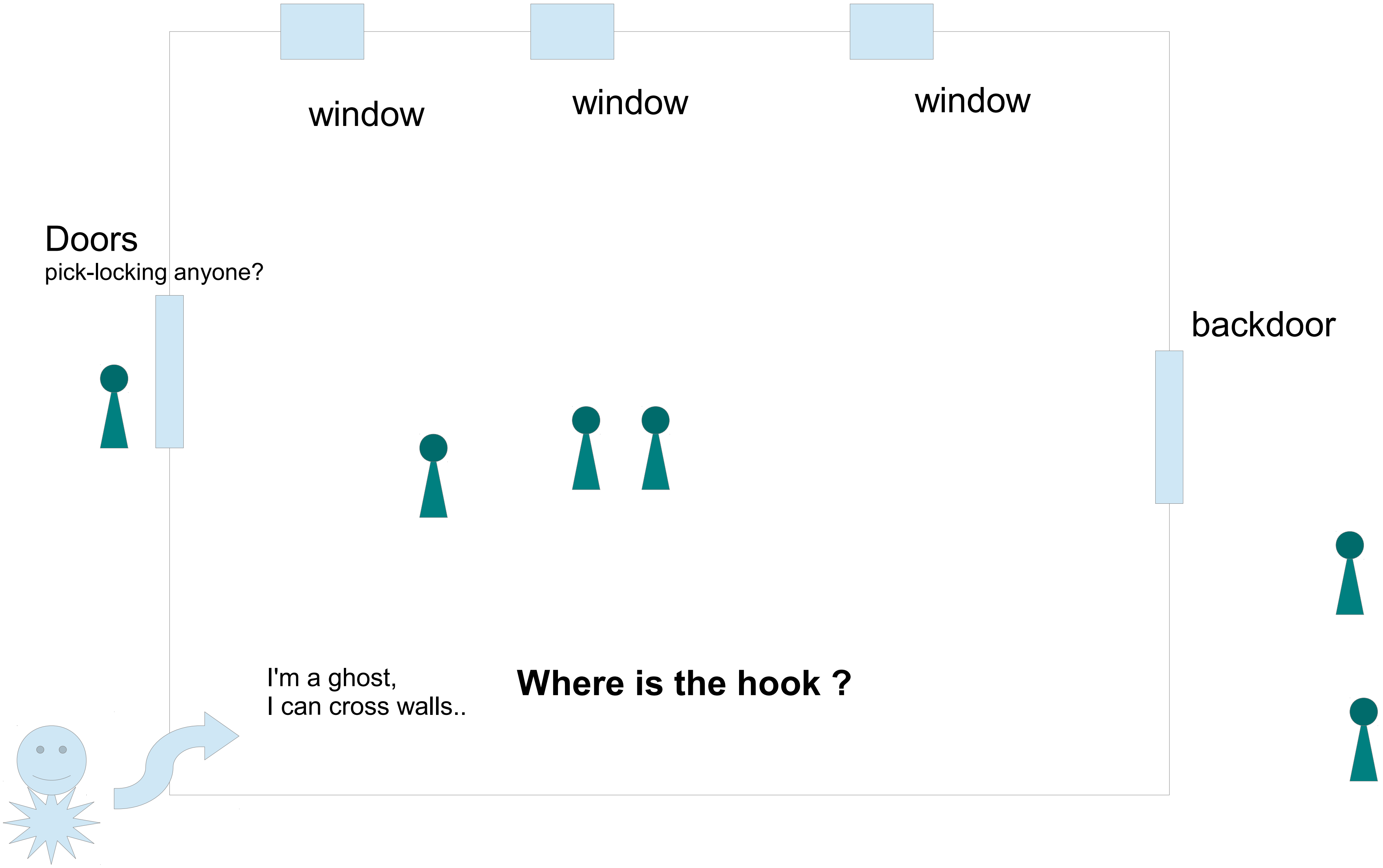
Information flow with hooks ?

- Ghosts ?



Information flow with hooks ?

- Ghosts ?



window

window

window

Doors
pick-locking anyone?

backdoor

I'm a ghost,
I can cross walls..

Where is the hook ?

Information flow with hooks ?

- **But** it's possible to catch incoherent status of course
  - Before there was 3 users inside, now there is 4 users.
- The incoherence will appears by keeping label informations on objects, and between two hooks.

Exiting #no_bullshit zone

What's next ? Security at KR season 3 ?..

- what are "technical mechanism" for security implementation ?
- It's called "hardened kernel"

→ ASLR, PaX, PIE/SSP, RELRO, toolchain, …

→ KR Season 3 ?

Linux Security Modules

-

Thanks hupstream for this event !
Kernel Recipes 2013