

# Authorization Schemes & Intrusion Tolerance for Internet Applications

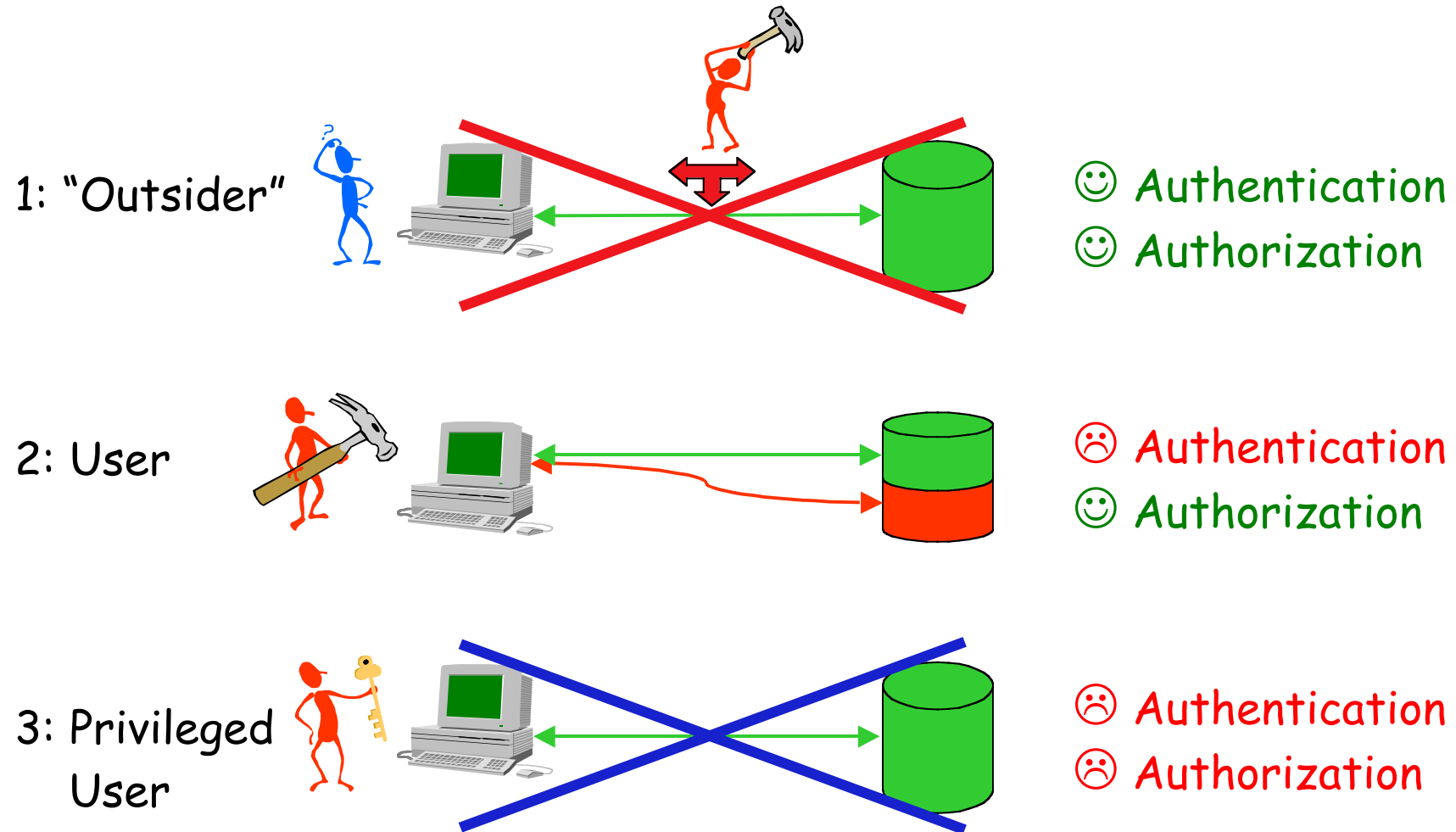
---



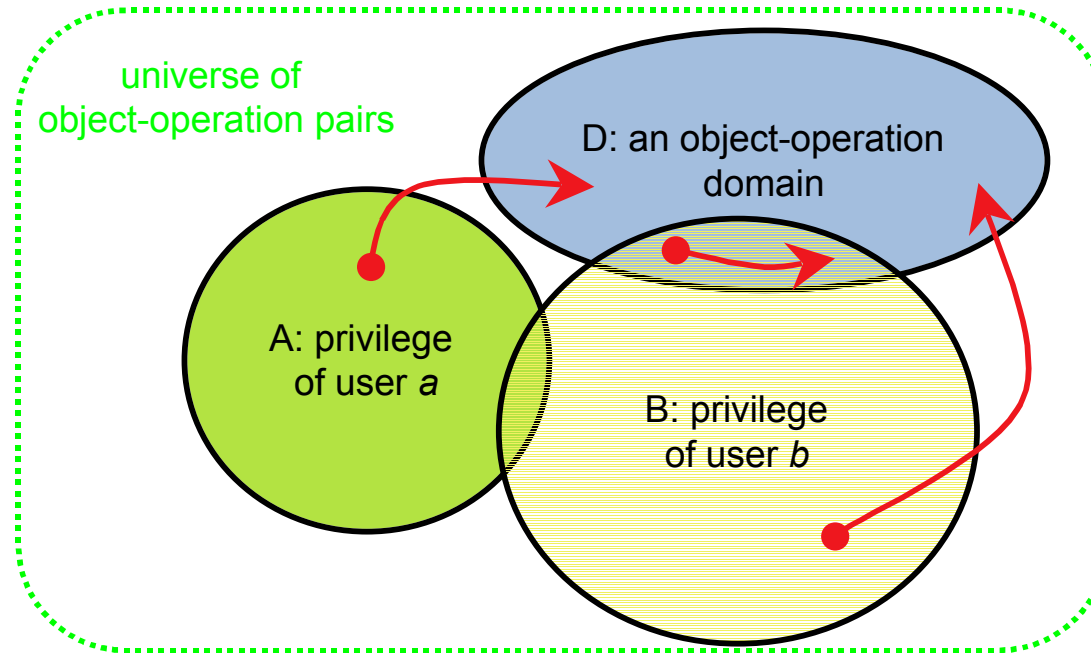
Yves Deswarte  
LAAS-CNRS



# Who are the intruders?



# Outsiders or Insiders: Privilege



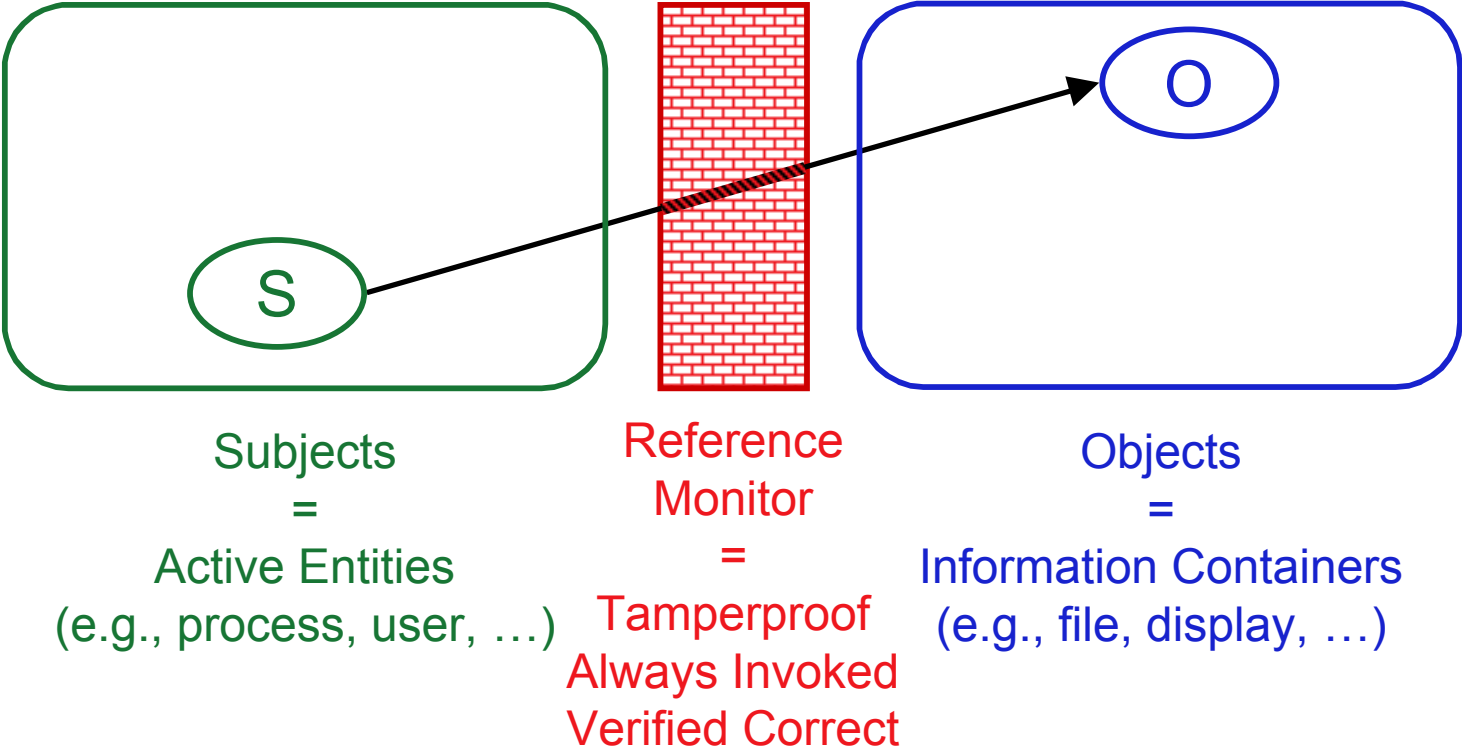
- ❖ **Theft of privilege:** unauthorized increase in privilege
- ❖ **Abuse of privilege:** improper use of authorized operations
- ❖ **Outsider:** current privilege does not intersect considered domain
- ❖ **Insider:** current privilege intersects considered domain

# Authorization

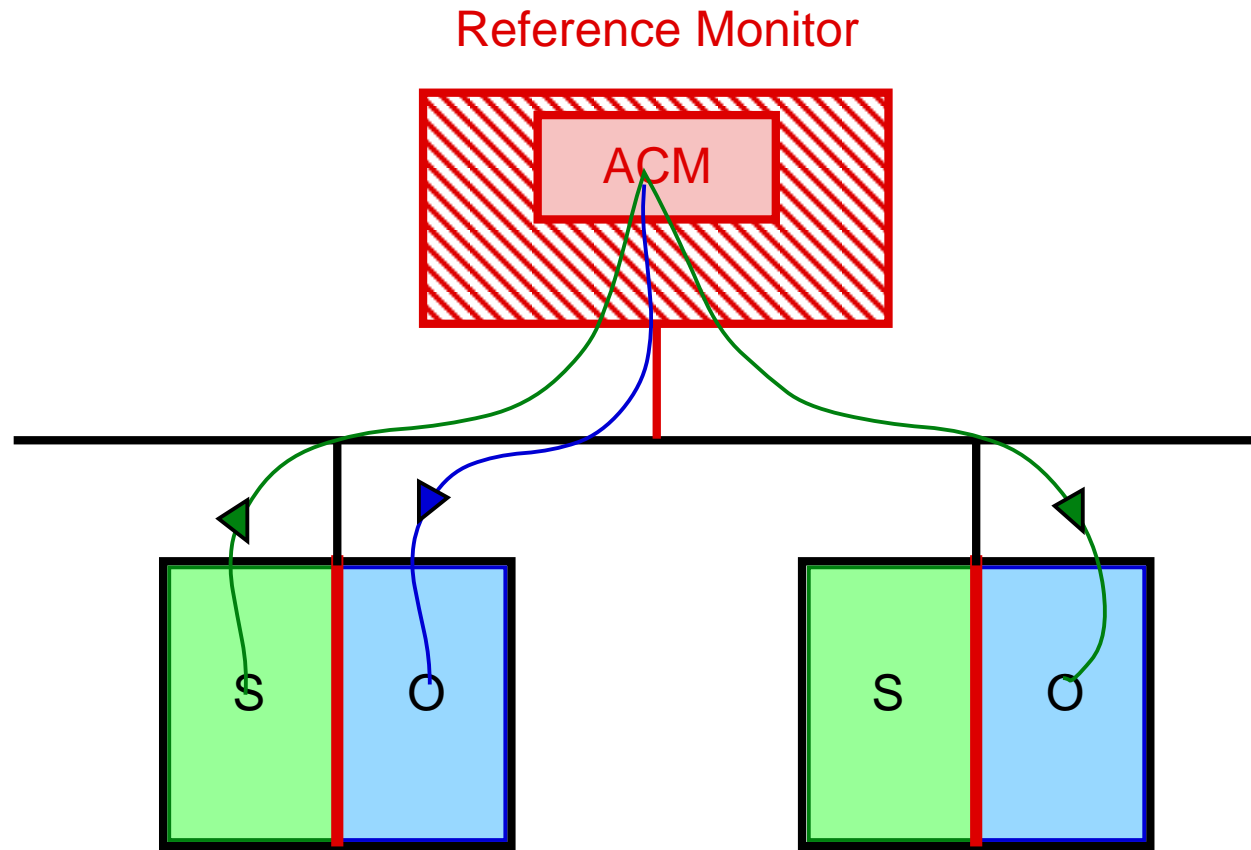


- ❖ Contributes to protection:
  - Error detection/confinement
  - Intrusion prevention/confinement
  
- ❖ For Internet applications:
  - More flexible than “client-server” paradigm
  - Contributes to privacy:  
personal information is disclosed only on a  
“need-to-know” basis

# Authorization: reference monitor



# Distributed Authorization ? (1)

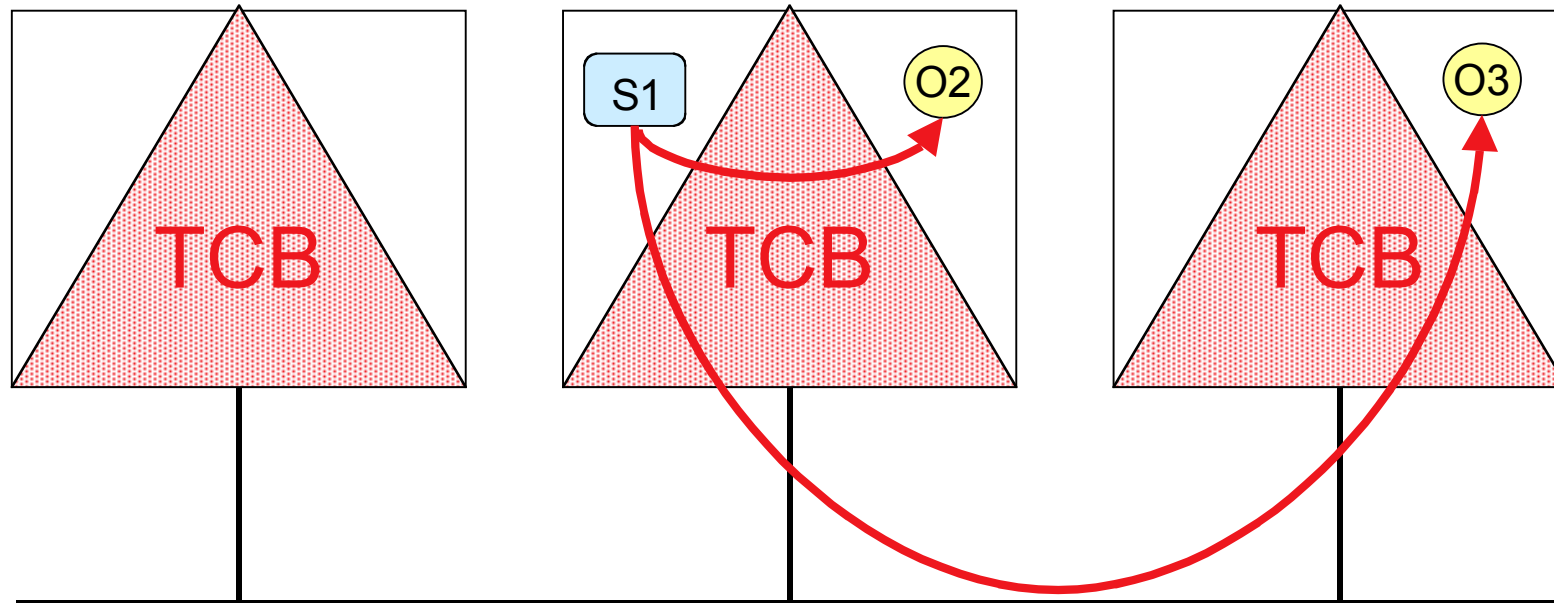


- ☺ small trusted area, easy administration
- ☹ bottleneck, single-point-of-failure

# Distributed Authorisation ? (2)



Red Book (TNI)

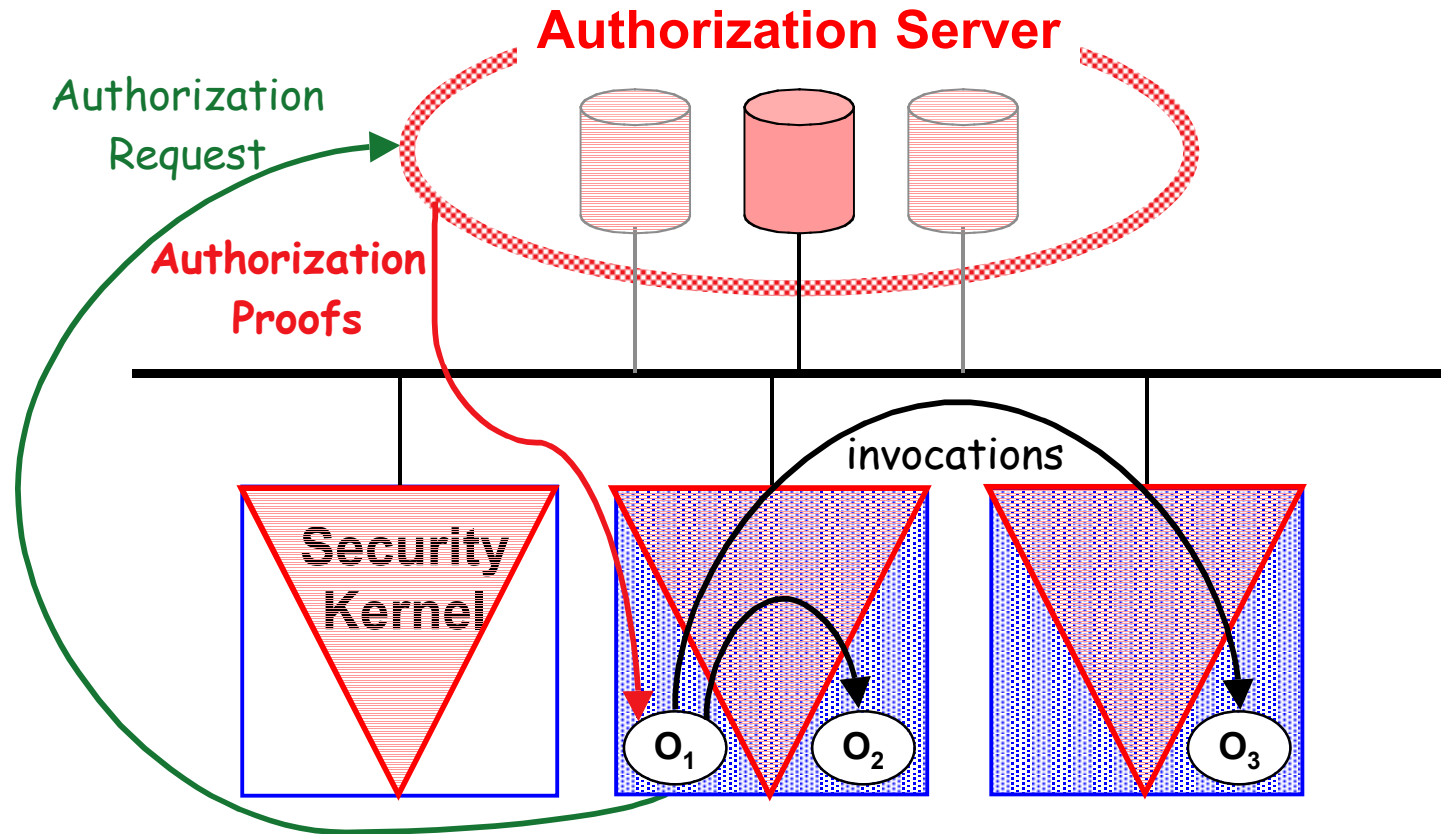


- ☺ No bottleneck, no single-point-of-accidental-failure
- ☹ Mutual trust between TCBs, consistency?

# Authorization Scheme for DOOS



[Nicomette & Deswarte, 1997]





# MAFTIA Authorization Server



Makes use of MAFTIA middleware:

- ❖ Non-confidential information is replicated  
(atomic multicast)
- ❖ Confidential information is shared securely  
(threshold crypto)
- ❖ Global consensus is achieved  
(majority voting / Byzantine agreement)
- ❖ Authorization proofs are distributed to local  
reference monitors  
(threshold signatures)

# Local protection

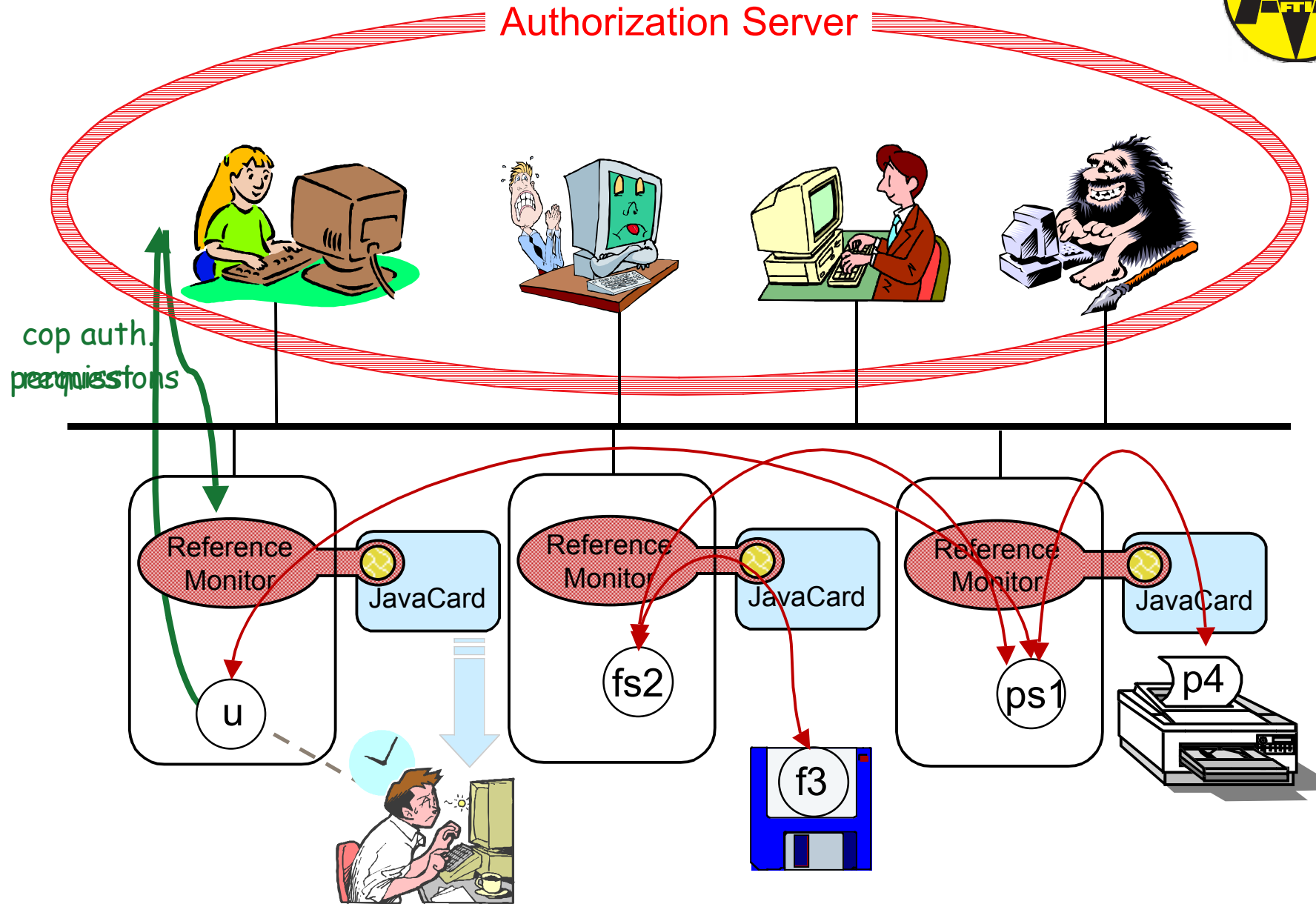
---



- ❖ Internet applications: heterogeneous platforms => **no modification** of user workstations or even servers
- ❖ => no trusted security kernel, but JVM
- ❖ Local Reference Monitor =
  - A local dispatcher (not trusted)
  - A JavaCard -> security kernel



# Authorization Server



# Security properties



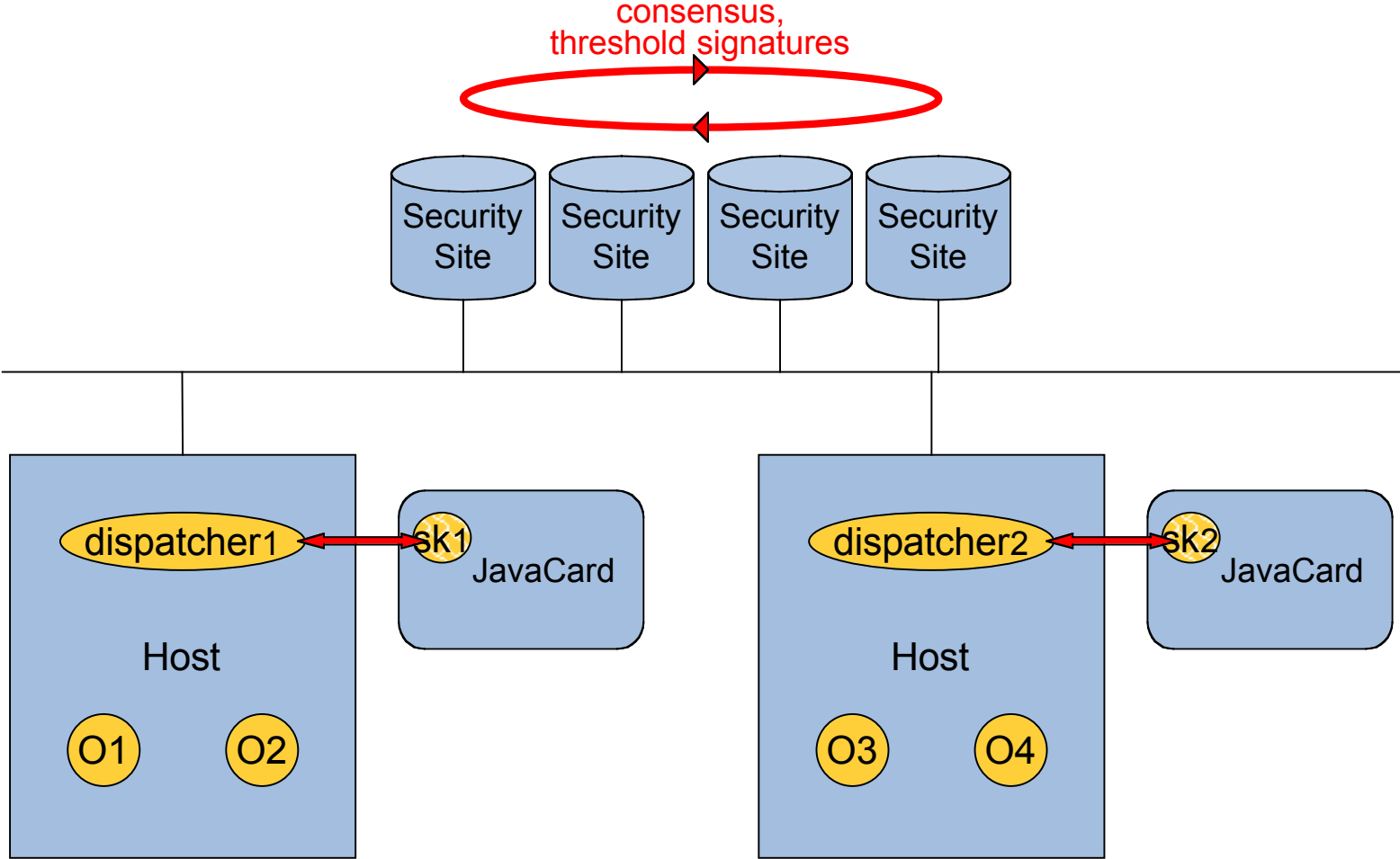
## ❖ Authorization server:

- AS1: The AS generates only valid authorization proofs
- AS2: It is not possible to prevent AS from generating valid authorization proofs

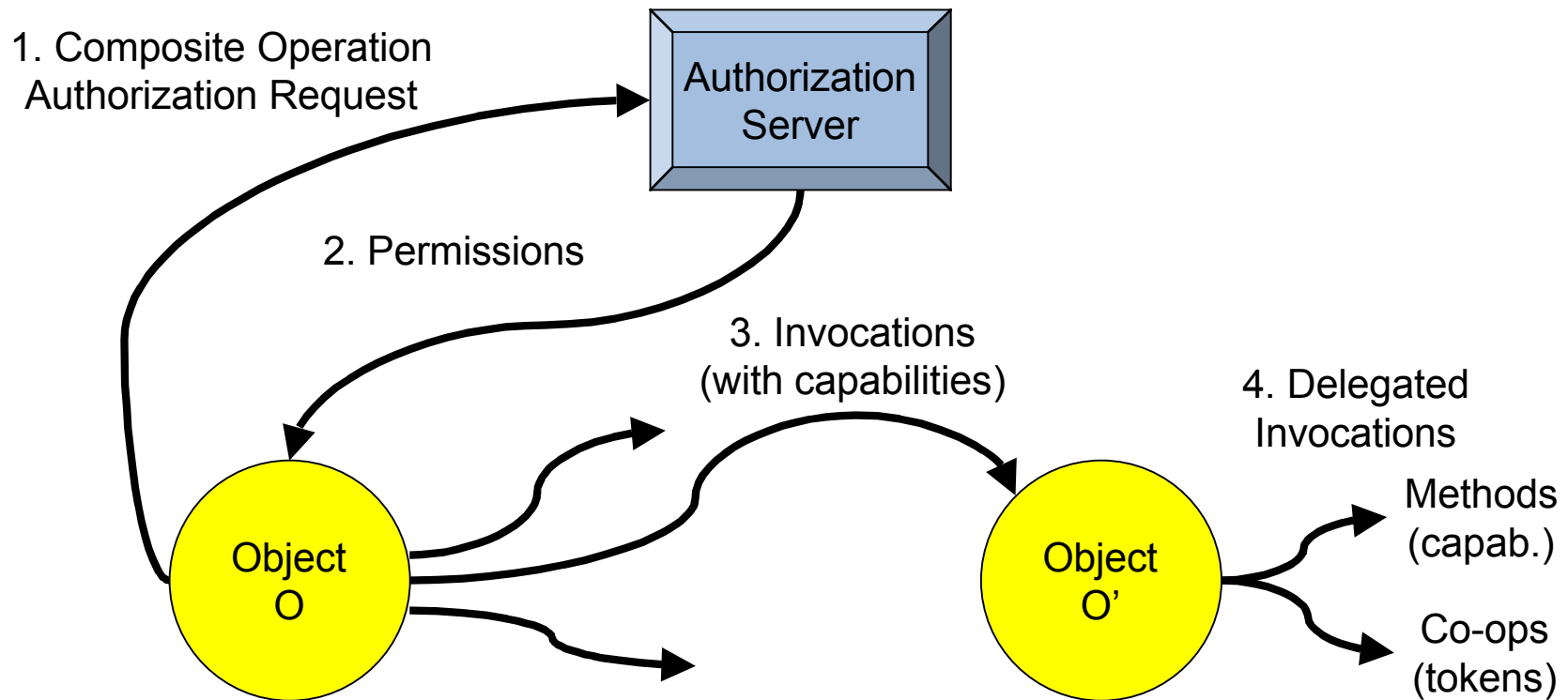
## ❖ Local reference monitors

- RM1: Only valid operations will be executed on a non-faulty host
- RM2: It is not possible to prevent non-faulty hosts from executing valid operations
  - Assumption1: no network denial-of-service
  - Assumption2: Java Card tamperproof

# Architecture



# Permissions



$$\text{Permissions}(O) = \langle \{\text{Perm}(O, O'.m)^*; \text{Perm}(O, \text{cop})^*\} \rangle_{SKas}$$

$$\text{Perm}(O, O'.m) = \{O; O'.m(\text{parC}); \text{cap}(O; O'.m(\text{parC}))[\text{vouch}(O'.m)]\}$$

$$\text{Perm}(O, \text{cop}) = \{O; \text{cop}(\text{parC}); \text{token}(O; \text{cop}(\text{parC}))\}$$

$$\text{cap}(O; O'.m(\text{parC})) = \langle \{O; O'.m; \text{parC}; \text{nonce}\} \rangle_{SKas, PKhost(O')}$$

$$\text{vouch}(O'.m) = \langle \{\text{Perm}(O', O''.m')^*; \text{Perm}(O', \text{cop})^*\} \rangle_{SKas}$$

$$\text{token}(O; \text{cop}(\text{parC})) = \langle \{O; \text{cop}; \text{parC}; \text{nonce}\} \rangle_{SKas, PKas}$$

# Cop Authorization Checks



**Symbolic rights:** corresponding to the authorization for an object to execute composite operations

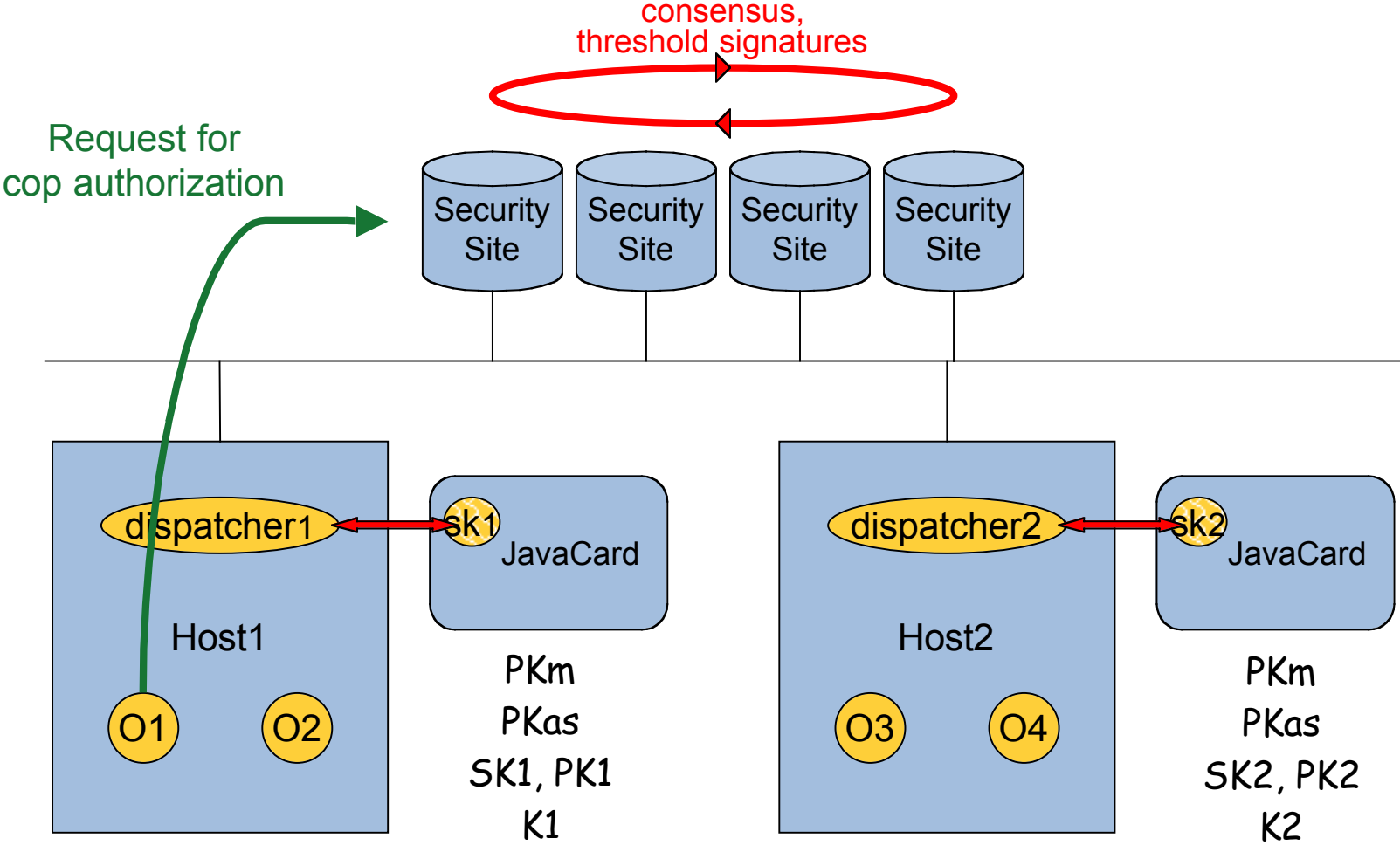
(a simple method execution is a particular case of cop)

	ps1	fs2	f 3	p 4
u			PF(this,PRINTER)	PF(FILE,this)
ps1				print
fs2				

**Symbolic right rules:** to check authorization for composite operations

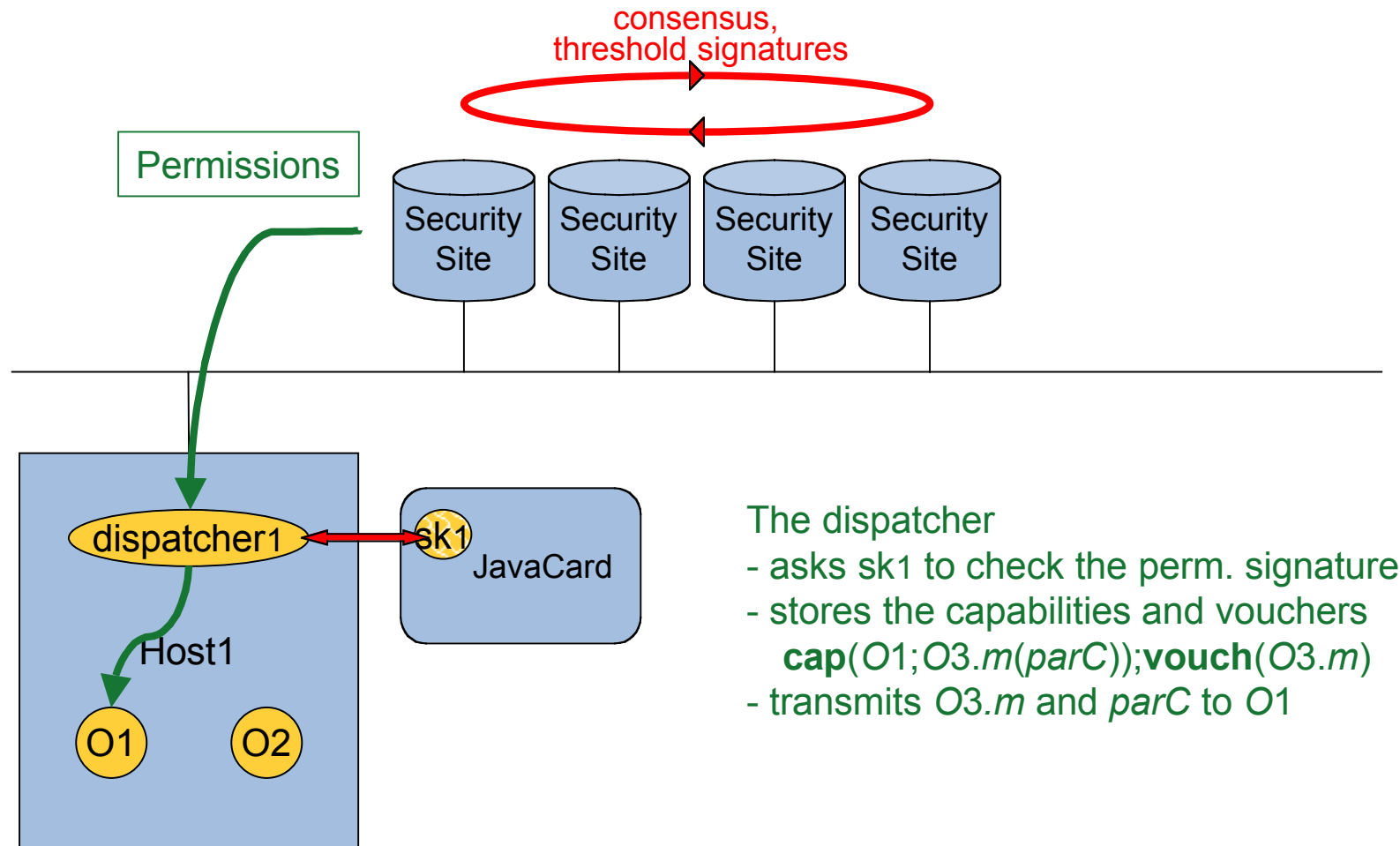
**Permission creation rules:** to generate permissions (capabilities and *vouchers*, tokens) to enable all methods executions

# Architecture





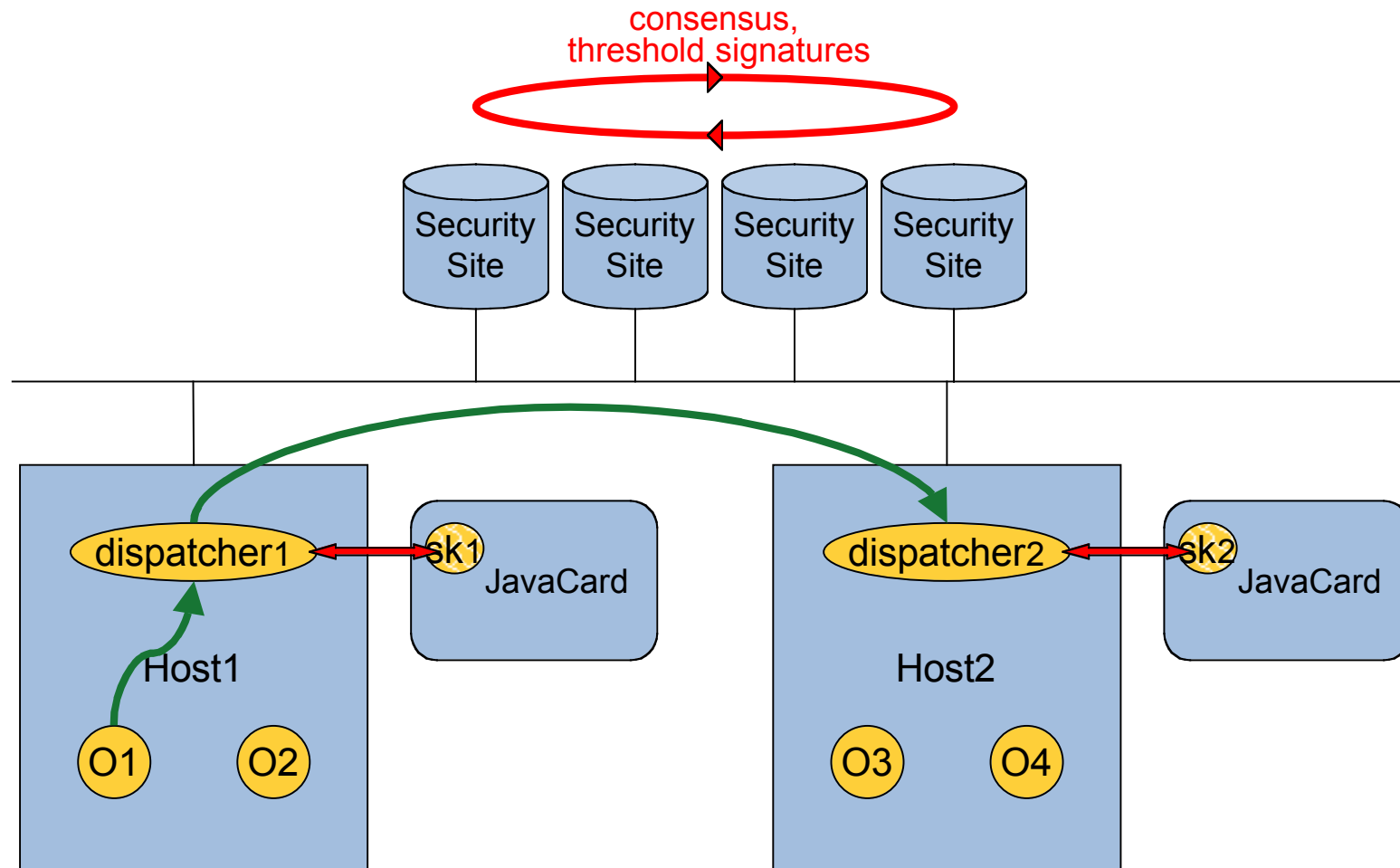
# Architecture



## The dispatcher

- asks sk1 to check the perm. signature
- stores the capabilities and vouchers  
 $\text{cap}(O1; O3.m(\text{parC})); \text{vouch}(O3.m)$
- transmits  $O3.m$  and  $\text{parC}$  to  $O1$

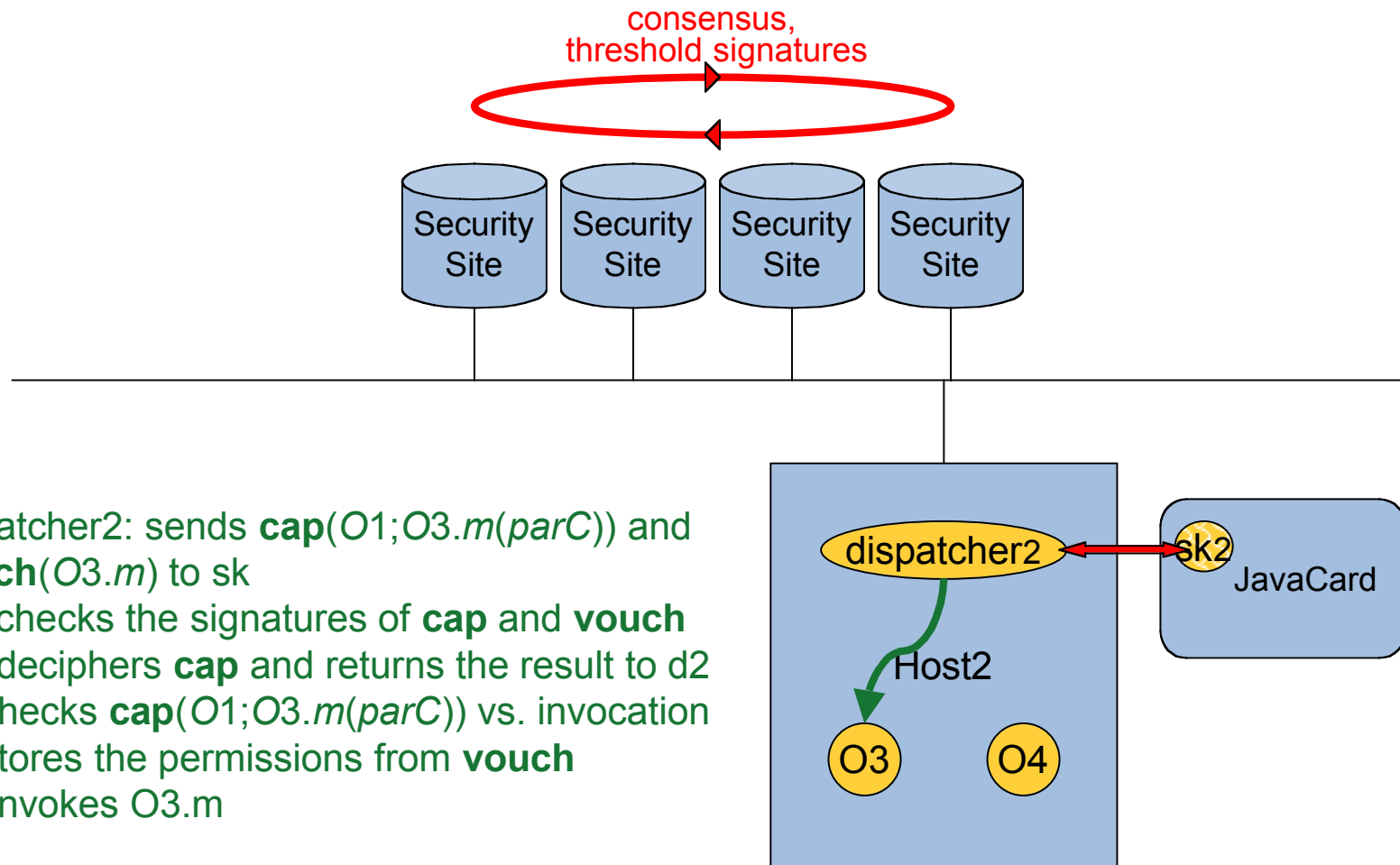
# Architecture



O1 invokes O3.m with {parameters}

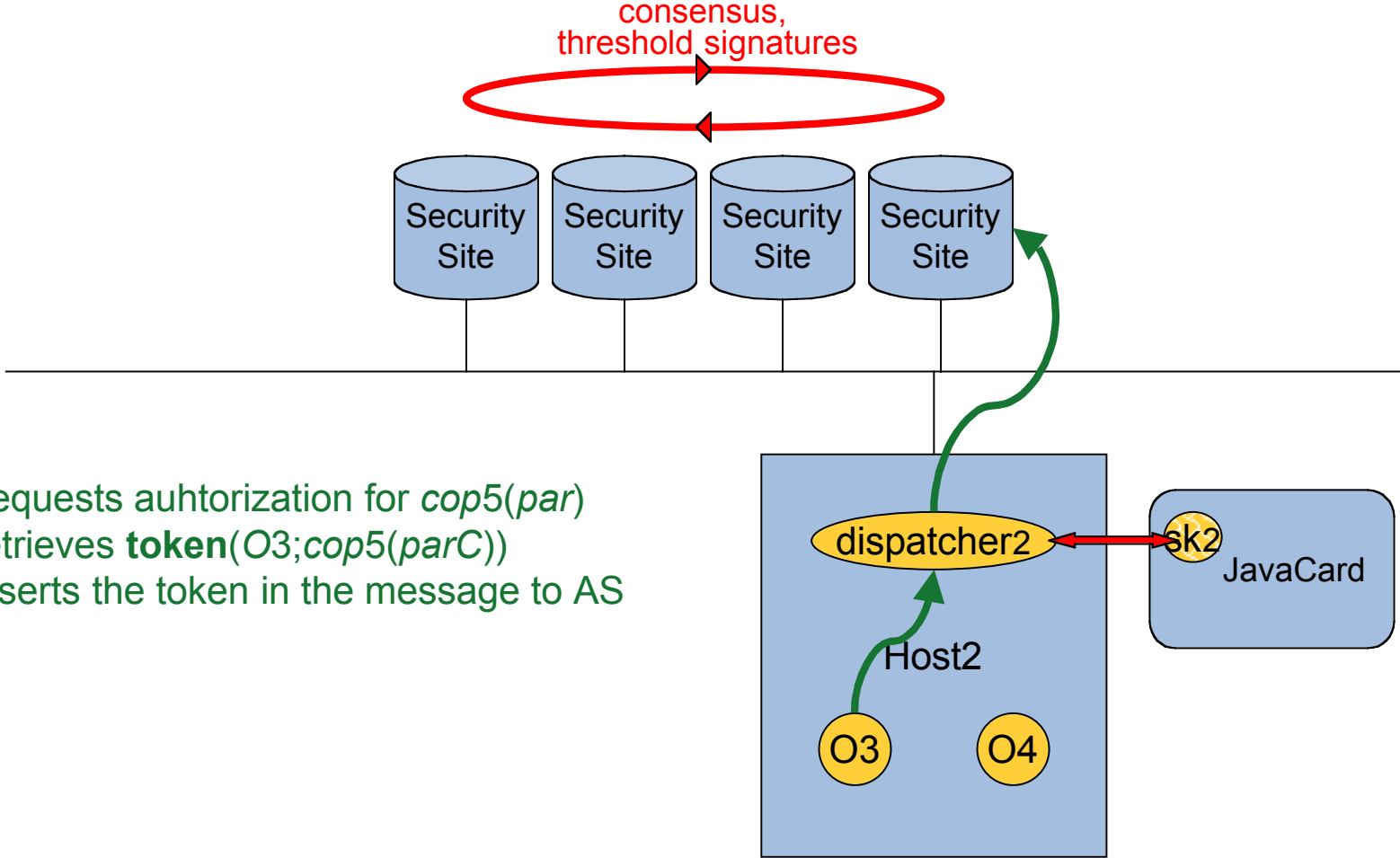
dispatcher1 retrieves  $\{\text{cap}(O1;O3.m(\text{parC}));\text{vouch}(O3.m)\}$ , inserts it in the message to disp.2

# Architecture



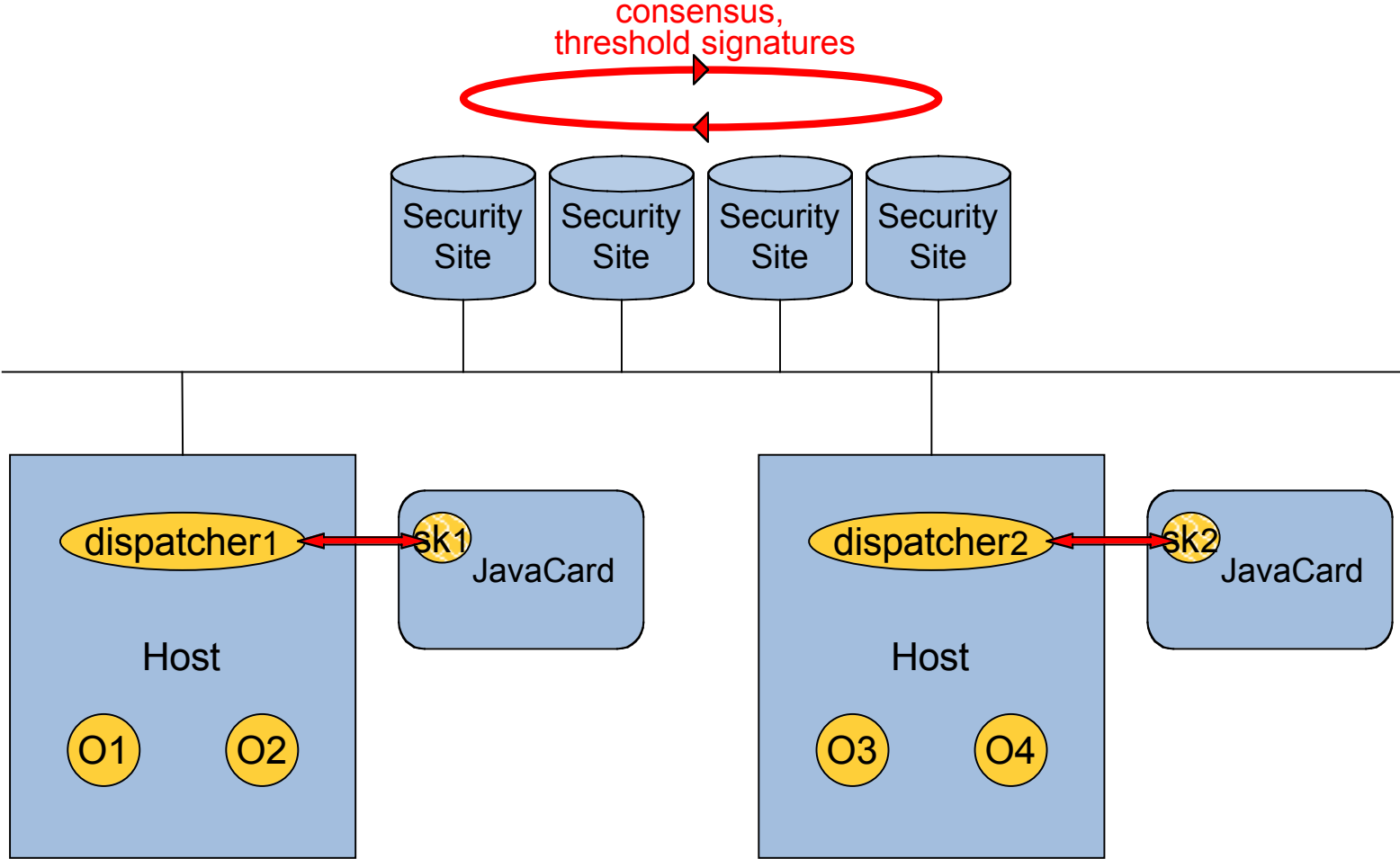
- dispatcher2: sends **cap**( $O1;O3.m(parC)$ ) and **vouch**( $O3.m$ ) to sk
- sk2 checks the signatures of **cap** and **vouch**
- sk2 deciphers **cap** and returns the result to d2
- d2 checks **cap**( $O1;O3.m(parC)$ ) vs. invocation
- d2 stores the permissions from **vouch**
- d2 invokes  $O3.m$

# Architecture

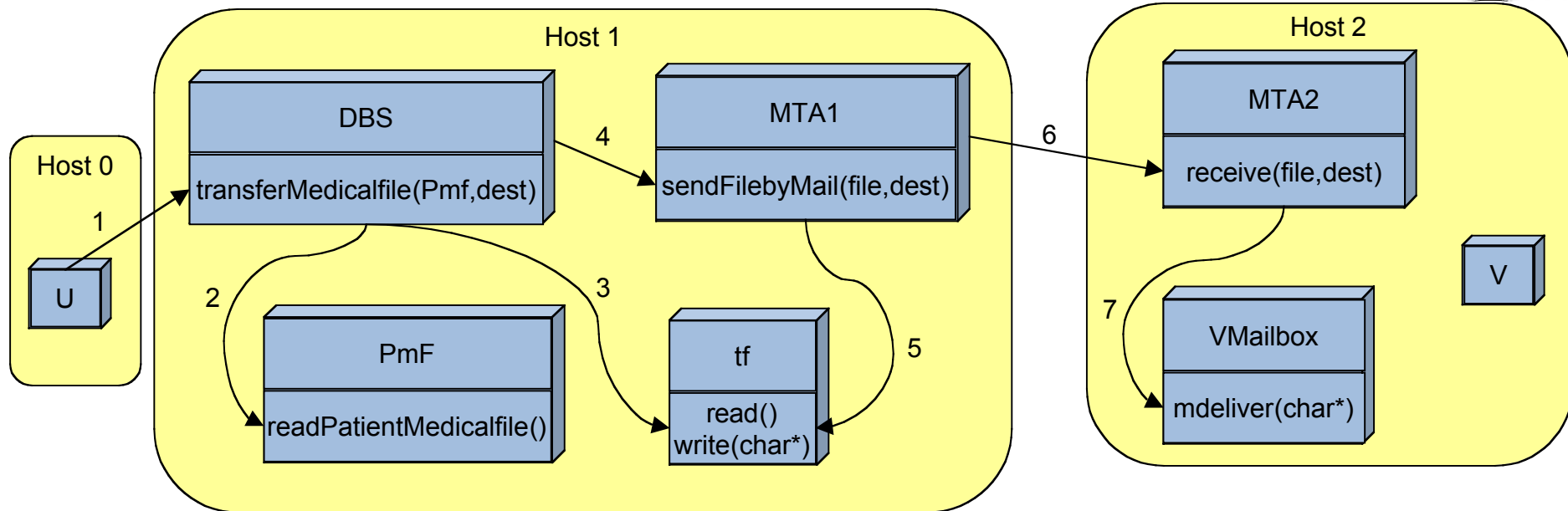


- O3 requests authorization for *cop5(par)*
- d2 retrieves **token**(O3;cop5(parC))
- d2 inserts the token in the message to AS

# Architecture



# Example:

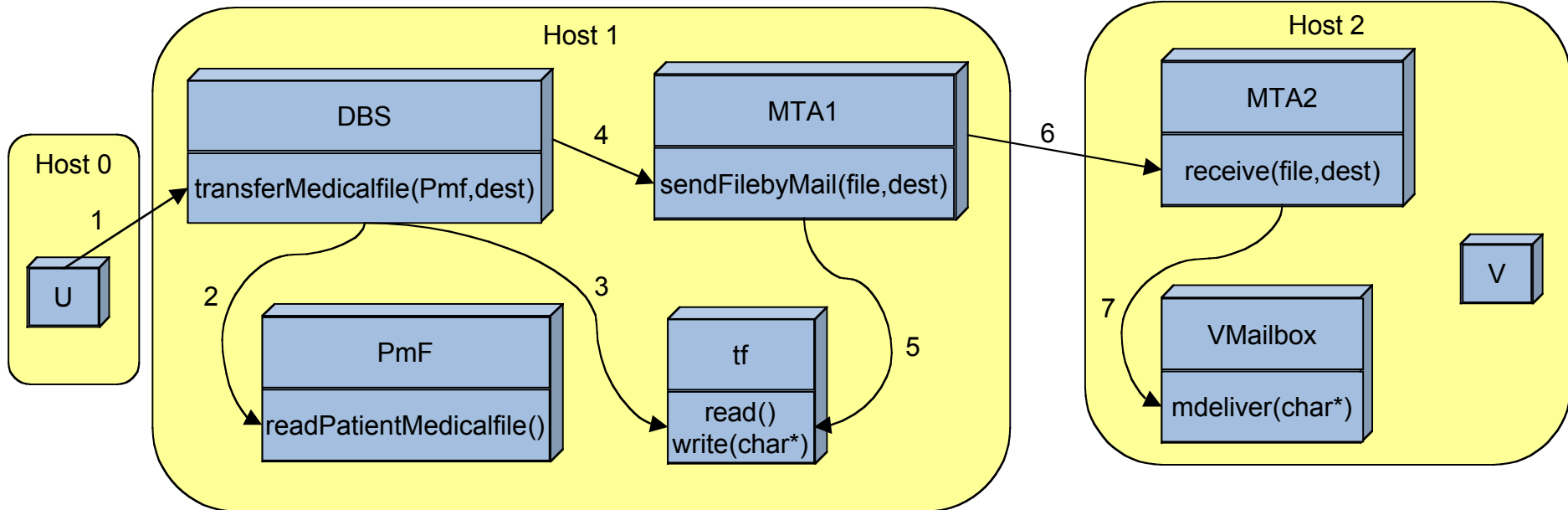


$U \rightarrow AS : \text{Request}(\text{SendPatientMedicalFile}(Pmf_1, V))$

	...	HCP Role	$\{Pmf(U)\}$
$U$		$SPmf(\{Pmf(U)\}, this)$	$read, write(*), SPmf(this, HCPRole)$
$DBS$	...		
...			

$AS \rightarrow D_0 : \left\{ U; DBS.transferPatientMedicalfile(Pmf_1, V); \text{Cap}(U; DBS.transferPatientMedicalfile(Pmf_1, V)); \text{Vouch}(DBS.transferPatientMedicalfile) \right\}$

# Example:

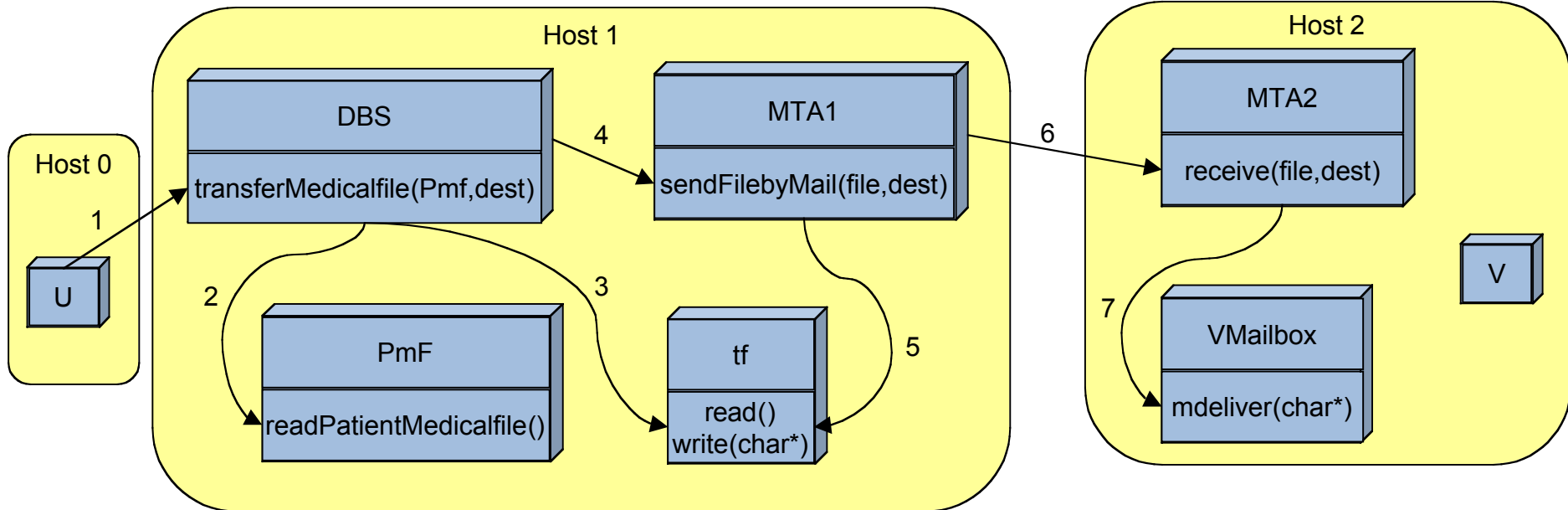


$$U \xrightarrow{1} DBS : \left\{ \begin{array}{l} \mathbf{RMI}(DBS.transferPatientMedicalfile(Pmf_1, V)); \\ \mathbf{Cap}(U; DBS.transferPatientMedicalfile(Pmf_1, V)); \mathbf{Vouch}(DBS.transferPatientMedicalfile) \end{array} \right\}$$

$$D_1 \rightarrow JC_1 : \left\{ \mathbf{Cap}(U; DBS.transferPatientMedicalfile(Pmf_1, V)); \mathbf{Vouch}(DBS.transferPatientMedicalfile) \right\}$$

$$\mathbf{Vouch}(DBS...) = \left\{ \begin{array}{l} DBS; Pmf_1.readPatientMedicalfile; \mathbf{Cap}(DBS; Pmf_1.readPatientMedicalfile); \\ DBS; MTA1.sendFilebyMail(*, V); \mathbf{Cap}(DBS; MTA1.sendFilebyMail(*, V)); \\ \mathbf{Vouch}(MTA1.sendFilebyMail) \end{array} \right\}$$

# Example:



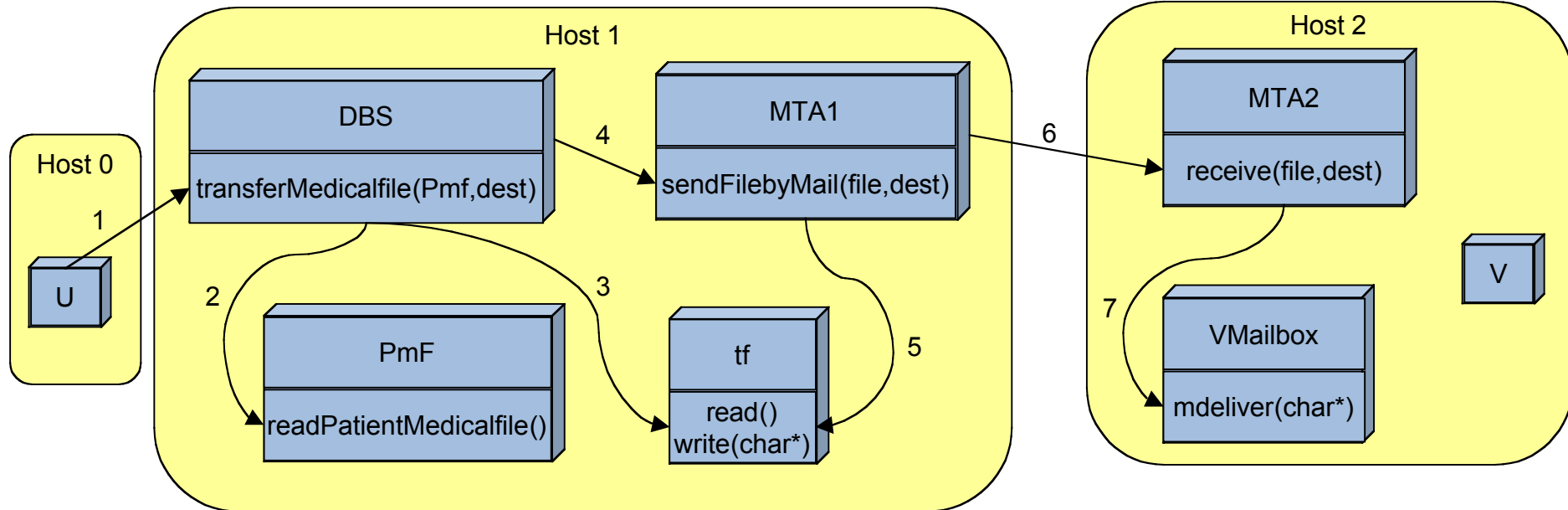
$$DBS \stackrel{2}{\mapsto} Pmf_1 : \{ \mathbf{RMI}(Pmf_1.readPatientMedicalfile( )); \mathbf{Cap}(DBS;Pmf_1.readPatientMedicalfile) \}$$

$$DBS \stackrel{3}{\mapsto} tf : \{ \mathbf{RMI}(tf.write(\langle \text{content of } Pmf_1 \rangle)) \}$$

$$DBS \stackrel{4}{\mapsto} MTA1 : \left\{ \begin{array}{l} \mathbf{RMI}(DBS.sendFilebyMail(tf,V)); \\ \mathbf{Cap}(DBS;MTA1.sendFilebyMail(*,V)); \mathbf{Vouch}(MTA1.sendFilebyMail) \end{array} \right\}$$



# Example:

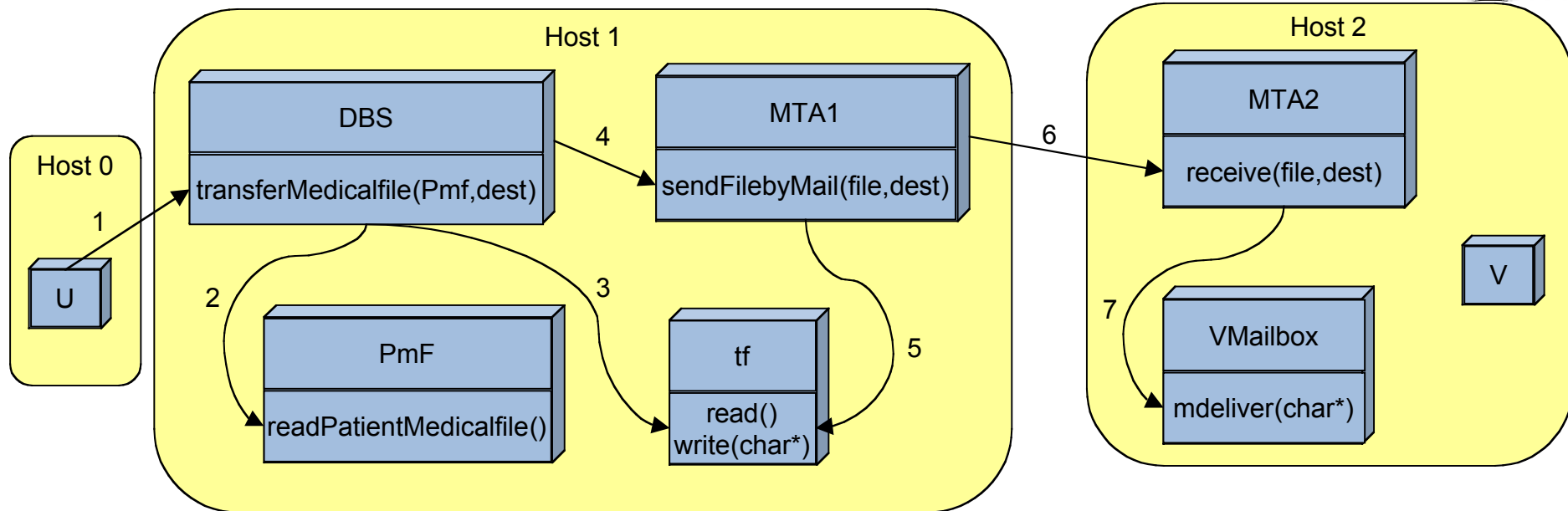


$MTA1 \rightarrow AS : \text{Request}(\text{DeliverFilebyMail}(*,V)); \text{Token}(MTA1; \text{DeliverFilebyMail}(*,V))$

$AS \rightarrow D_1 : \{ MTA1; MTA2.receive(*,V); \text{Cap}(MTA1; MTA2.receive(*,V)); \text{Vouch}(MTA2.receive) \}$

$MTA1 \xrightarrow{6} MTA2 : \left\{ \begin{array}{l} \text{RMI}(MTA2.receive(\langle \text{content of } tf \rangle, V)); \\ \text{Cap}(MTA1; MTA2.receive(*,V)); \text{Vouch}(MTA2.receive) \end{array} \right\}$

# Example:



$$D_2 \rightarrow JC_2 : \{ \mathbf{Cap}(MTA1; MTA2.receive(*, V)); \mathbf{Vouch}(MTA2.receive) \}$$

$$MTA2 \stackrel{7}{\mapsto} VMailbox : \{ \mathbf{RMI}(VMailBox.mdeliver(\langle \text{content of } tf \rangle)); \mathbf{Cap}(MTA2; VMailbox.mdeliver(*)) \}$$

<http://www.research.ec.org/maftia/>

