UNIVERSITY
OF TRENTO - Italy

secure
CHANGE

Autonomous and yet Secure Evolution for Smart Cards Applications

Fabio Massacci
Joint work with O. Gadyatskaya
DISI, University of Trento

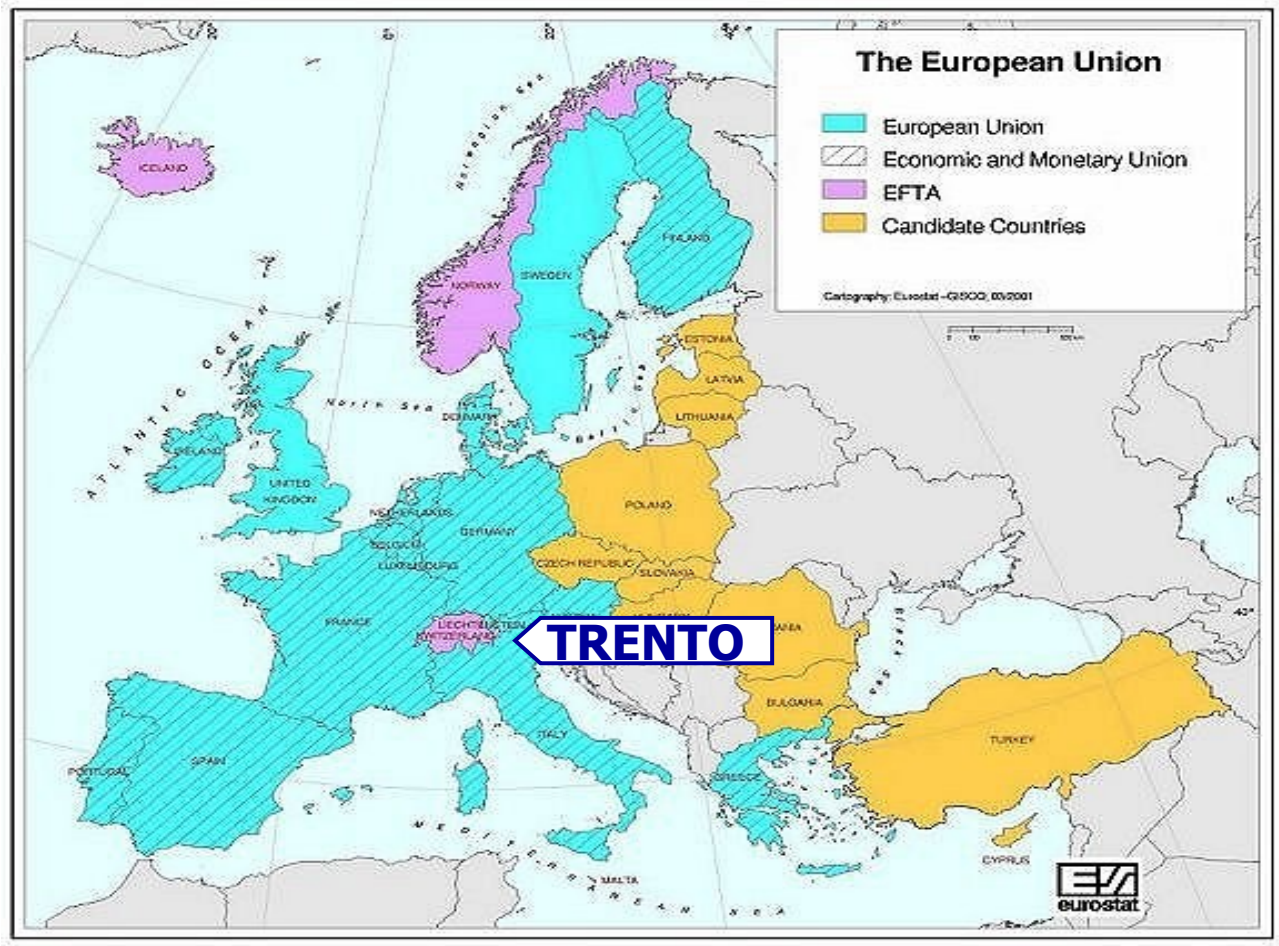
The talk plan

[redacted]

secure CHANGE

- **Where's Trento?**
- **The rara avis of multi-application smart-cards**
- **Security-by-Contract for smart cards**
- **A (thin) slice of theory**
- **A (larger) slice of engineering**
- **Open problems**

Trento in Space and Time



1962

- Institute of Social Science founded as locally funded Institution

1972

- Institute becomes private University

1982

- University becomes a state University with special autonomy

2001

- University becomes 1st in University Rankings

What do we do there?

- **Organizational Level Security**
  - Governance, Risk and Compliance (FM)
  - Security Requirements Engineering (FM,JM,PG)
- **System Security**
  - Run-time enforcement at ESB (FM,BC)
  - Browser Security (FM)
- **Mobile/Embedded Code Security-by-Contract**
  - Load-time security verification (FM)
  - Run-time information flow (BC)

The talk plan

[redacted]

secure CHANGE

- Where's Trento?
- **The rara avis of multi-application smart-cards**
- **Security-by-Contract for smart cards**
- **A (thin) slice of theory**
- **A (larger) slice of engineering**
- **Open problems**

Smart cards today

- **Modern computing devices**
- **Tamper-resistant security system**
- **Widely used**
- **But we have too many of them in our pockets**

Open Multi-application smart cards

[redacted]

secure CHANGE

- **Cards with multiple applets**
  - allow post-issuance evolution (add/remove/update)
  - from different stakeholders
  - Asynchronously
- **Interaction of applets on a single chip is natural:**
  - Applications may interact exchanging loyalty points, transferring money or sharing valuable information.
- **First paper I saw, I was a PhD Student 10yrs ago**
  - Information Flow Verification for Multi-Applications Smart Card.
  - The Air-France, Hertz example...

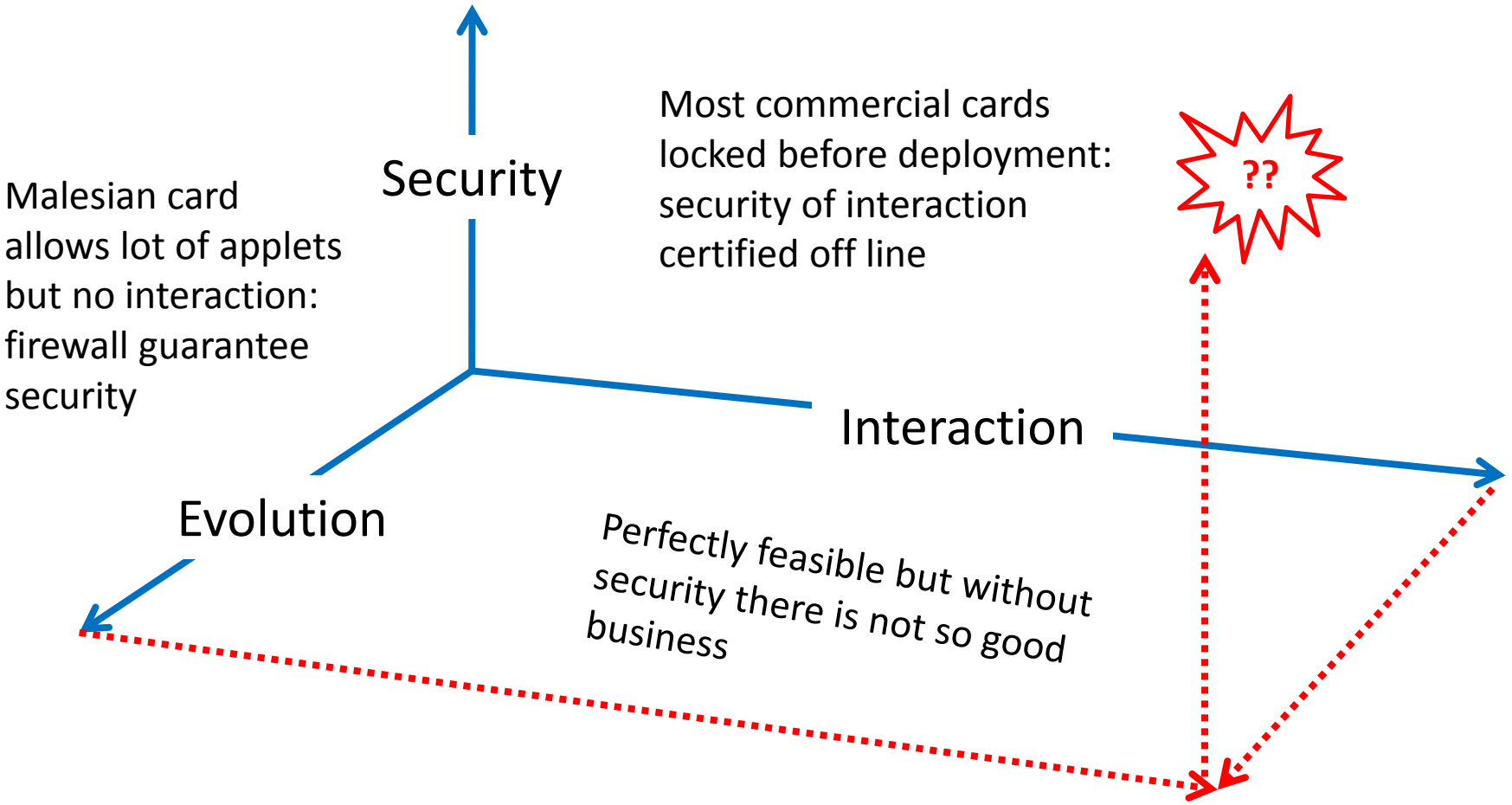
Java Card + GlobalPlatform

[redacted]

secure CHANGE

- **GlobalPlatform = Middleware for secure management of applets (with open specification)**
  - **Lots of smart cards deployed with GP**
- **GlobalPlatform and JavaCard specifications**
  - **support loading, update and un-loading of many applications on the fly and asynchronously**
  - **allow interactions among applications (through services implementing Shareable interface)**
- **Still/Yet/But…**
  - **We don't really see multi-application cards in the wild.**

What is there…



The usual evocative picture

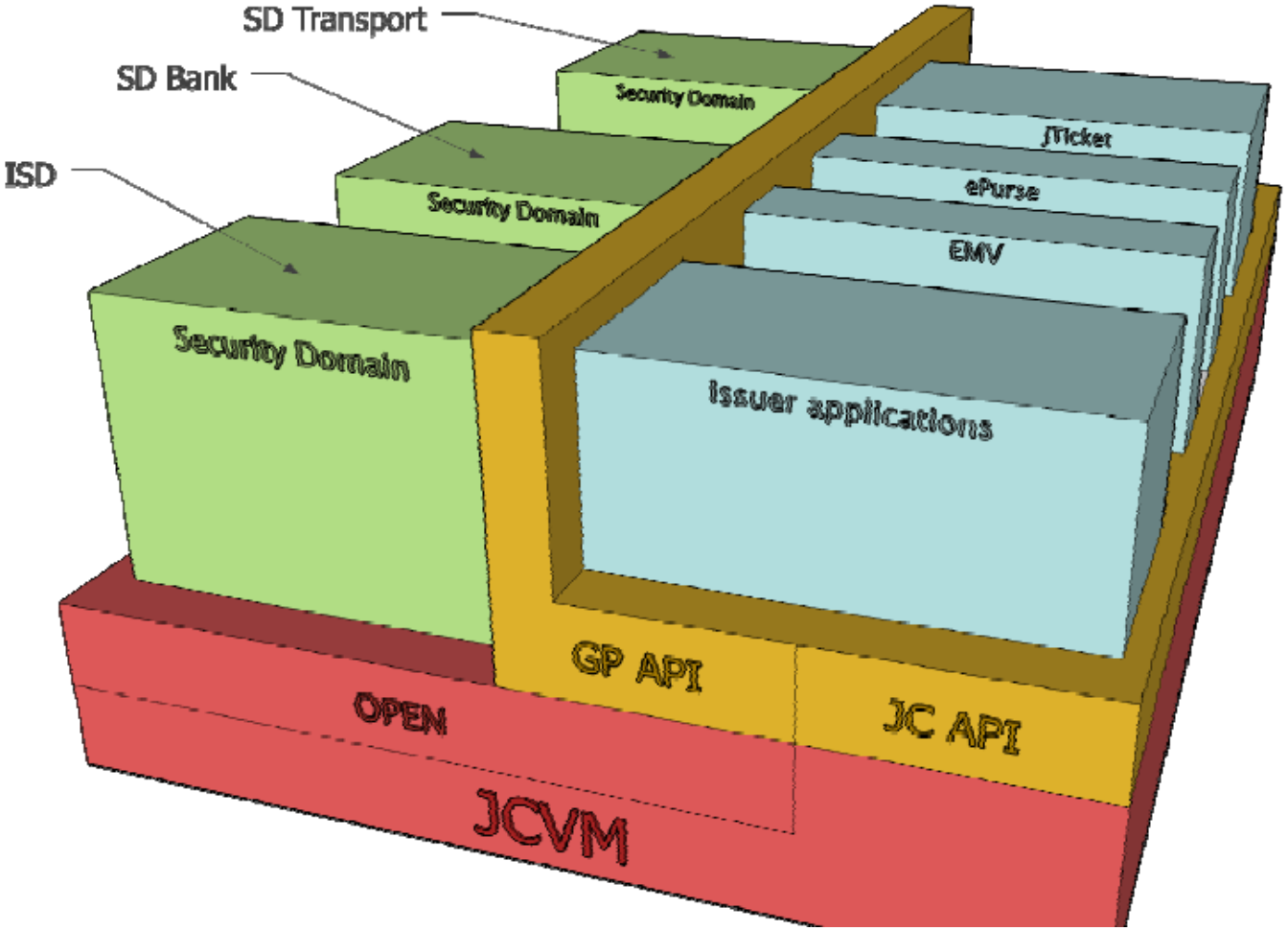
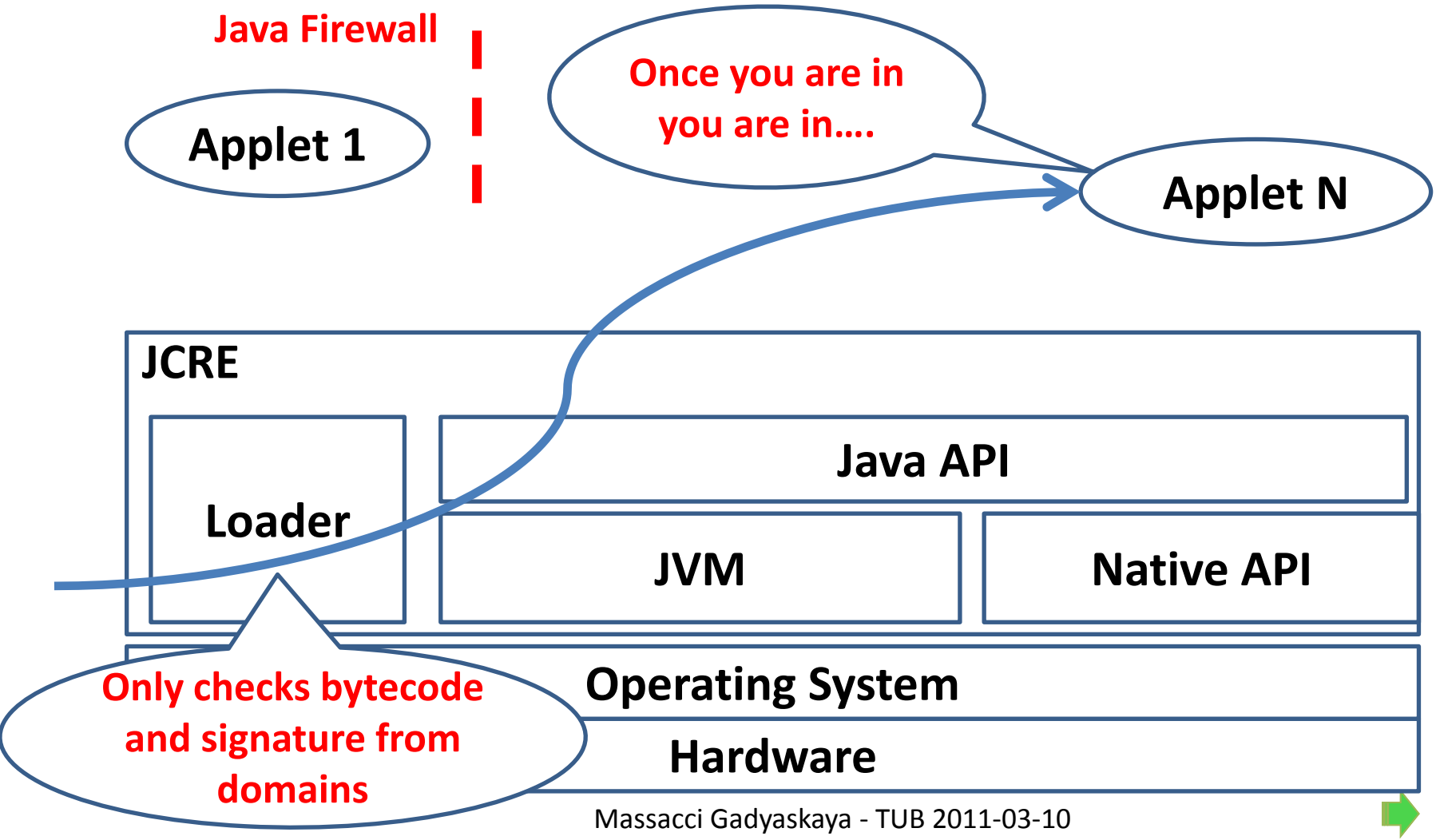


Image from D1.1 of SecureChange project

A more precise picture



How does Java Card firewall work?

[redacted]

secure CHANGE

- **Applets interact through firewall using shareable interfaces**
- **Application ePurse of Bank**
  - **offers a service transfer_money.**
  - **does a preliminary access control checking caller AIDs in a list**
- **Application jTicket of Transport**
  - **wants to use transfer_money of ePurse**
- **What happens**
  - **jTicket asks the firewall for a reference to transfer_money.**
  - **Firewall passes call to ePurse. If jTicket is in the list, ePurse will return a reference to transfer_money service.**
- **Consequences**
  - **jTicket got a reference → can use service from now on**
  - **ePurse wants to prevent jTicket use its service → must update itself**

→ Business Model of Multi-Application SC (E+I+S) not supported

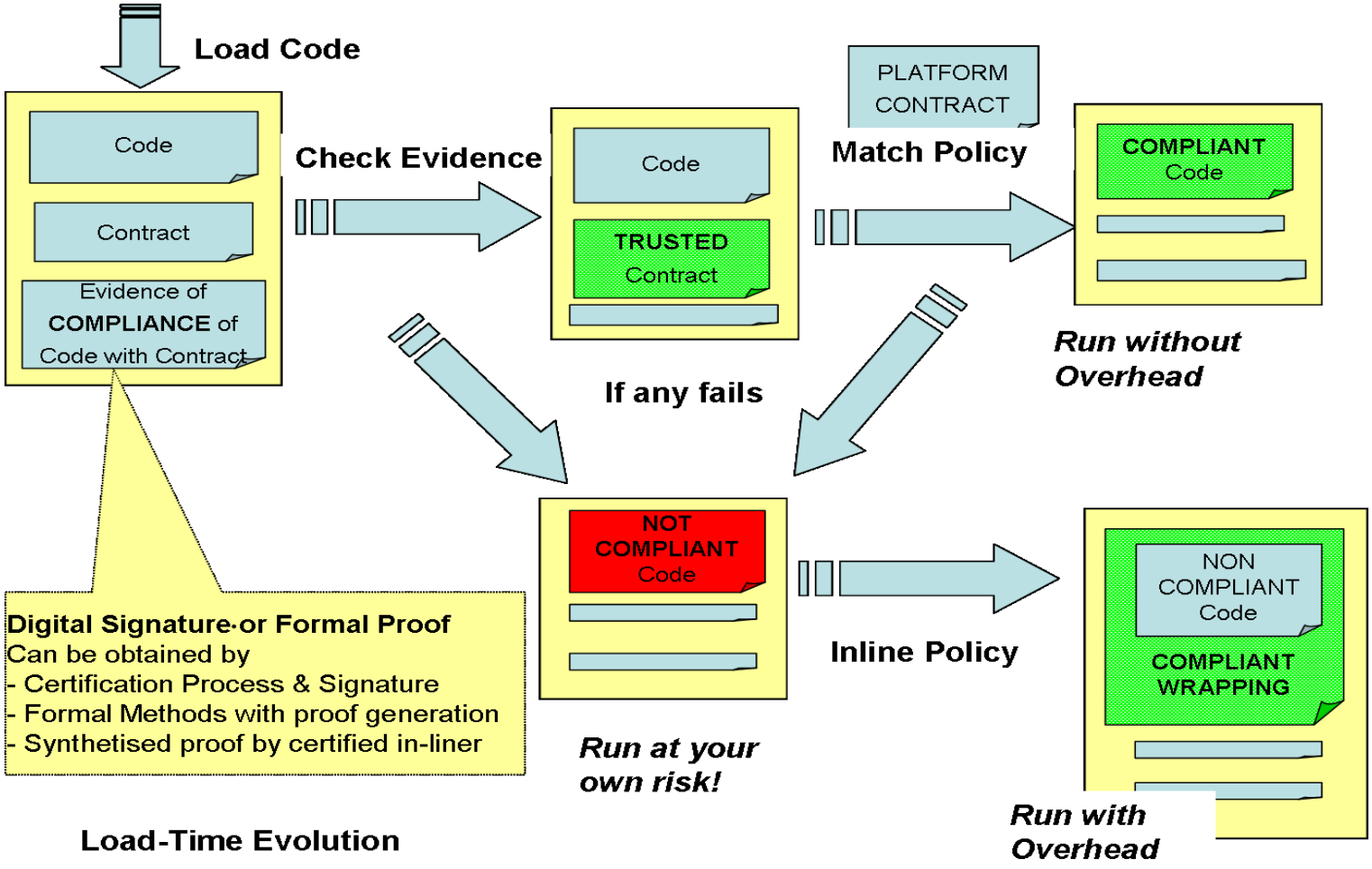
The talk plan

[redacted]

secure CHANGE

- Where's Trento?
- The rara avis of multi-application smart-cards
- **Security-by-Contract for smart cards**
- **A (thin) slice of theory**
- **A (larger) slice of engineering**
- **Open problems**

Security-by-Contract idea



SxC as Load-time verification

[redacted]

secure CHANGE

- **General idea of SxC for mobile devices:**
  - **Application has to be compliant with security policy of the device**
  - **Derived from PCC and MCC**
- **Well-tested for mobile platforms**
  - **Java & .NET implementation**
  - **Eu S3MS project**
  - **Many publication: JCS, JLAP, Comp. & Security, SCP, Elsevier IITR**
  - **Policy checker could even run a small model checker**
    - "allowed file.size > 1024Kb " vs "filesize < 512kb"
- **But here we have a problem**
  - **Who sets the policy of device?**
  - **"Clear" for mobiles: operator, manfacturer, user**

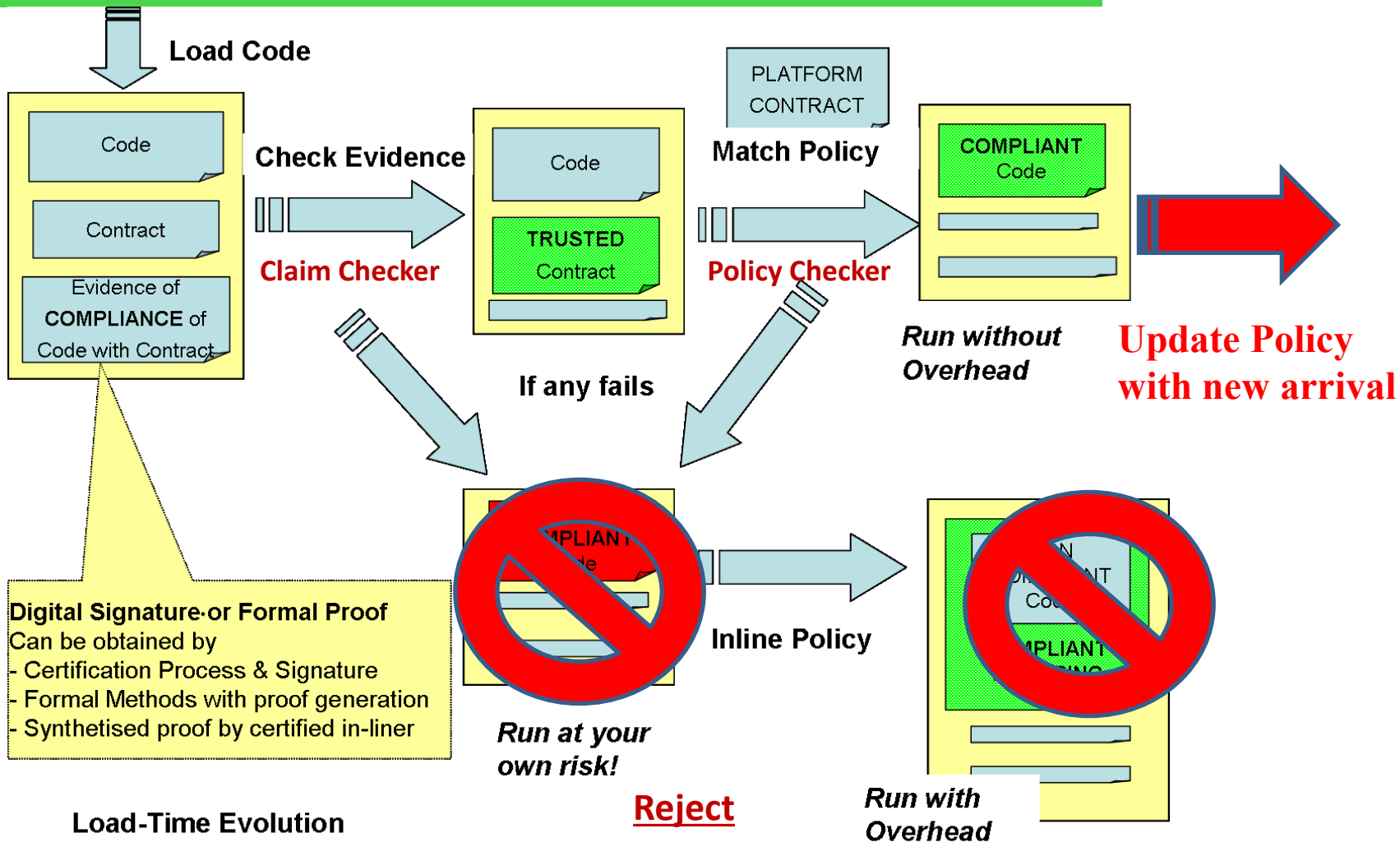
SxC for Smart Cards

[redacted]

secure CHANGE

- **Whose policy?**
  - **The union of the policies of all applets**
- **Broader Contract**
  - **Claims**
    - I may provide these shareable interfaces
    - I may call those methods from those interfaces
  - **Security Rules**
    - I can only be called by this Application/Package
  - **Functional Rules**
    - I need these methods from those interfaces

SxC workflow for smart cards



SxC Example

[redacted]

secure CHANGE

- **Already installed Applet ePurse with Contract:**

  - **Provides = {*transferMoney*}**
  - **Calls = {}**
  - **Sec.rules = {*transferMoney -> {jTicket*}}**
  - **Func.rules = {}**

- **Applet jTicket arrives with Contract:**
- **jTicket is loaded, cheked, and finally installed.**

  - **Provides = {ageDiscount, loyaltyDiscount}**
  - **Calls = {ePurse.transferMoney}**
  - **Sec.rules = { ageDiscount →{IDapplet), loyaltyDscount →{ePurse)}**
  - **Func.rules = {ePurse.transferMoney}**

- **Applet i-Travel arrives with Contract:**
- **i-Travel is rejected: load process is not committed**

  - **Provides = {}**
  - **Calls = {*ePurse.transferMoney*}**
  - **Sec.rules = {}**
  - **Func.rules ={}**

Formal Model of a JC Platform

[redacted]

secure CHANGE

***Platform* Θ =**

<Δ_A, Δ_S ,$\mathcal{A}$, shareable(), invoke(), sec.rules(), func.rules()>

- **Δ_A = domain of applications, Δ_S = domain of services**
- **$\mathcal{A} \subseteq \Delta_A$**
  - applets deployed (installed) on the platform
- **shareable(), invoke(): $\Delta_A \rightarrow p(\Delta_S)$**
  - Services offered by applet (resp. invoked by applet)
- **sec.rules(): $\Delta_A \times \Delta_S \rightarrow p(\Delta_A)$**
  - *For any applet and its services which applets can call it*
- **func.rules(): $\Delta_A \rightarrow p(\Delta_S)$**
  - *Services that must be present in order for the applet to function*

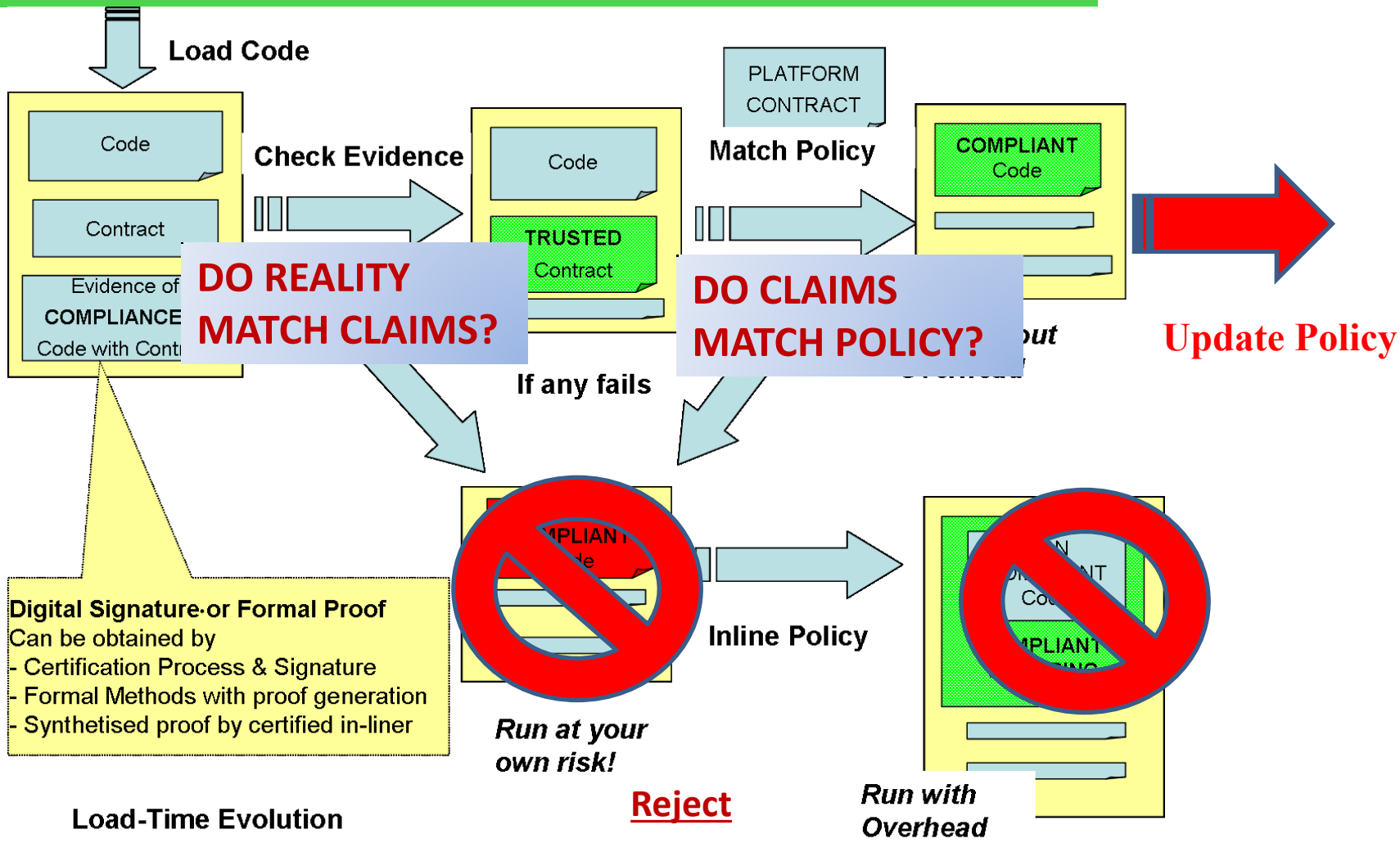
Why we use different names?

[redacted]

secure CHANGE

- **Platform has**
  - **Shareable(A) $\subseteq \Delta_S$ and invoke(A) $\subseteq \Delta_S$**
- **Contract has**
  - **Provides(A) $\subseteq \Delta_S$ and Calls(A) $\subseteq \Delta_S$**
- **Same difference between reality and claims**
  - **The first is reality, what really is there**
  - **The seconds are the claims, they might be honest but might also not correspond to truth**

SxC workflow for smart cards



The talk plan

[redacted]

secure CHANGE

- Where's Trento?
- The rara avis of multi-application smart-cards
- Security-by-Contract for smart cards
- **A (thin) slice of theory**
- **A (larger) slice of engineering**
- **Open problems**

Introducing evolution to the model

- **Let B be an application, an evolved platform Θ’ for B from a platform Θ is defined according to the next types of changes:**
  - **B is a new applet to be added to the platform,**
  - **old applet B is removed from the platform,**
  - **update of an installed applet B**
    - Add/remove of a service to shareable(B)
    - Add/remove of a service to invoke(B)
    - Add/remove of an access authorization to sec.rules(B)
    - Add/remove of a service to func.rules(B)

Checking Changes Incrementally

[redacted]

secure CHANGE

- **For each type of change the Claim Checker and the Policy checker should verify only the parts of the platform that are touched by changes.**
- **For new applet B:**
  - **Claim Checker has to verify that**
    - shareable(B)=ProvidesB
    - invoke(B)=CallsB
    - (or to extract shareable(B) and invoke(B) from the code and write these sets into the ContractB)
  - **The Policy Checker has to check that for all applets A ∈ A :**
    - if A.s ∈ CallsB then (s,B) ∈ sec.rules(A)
    - if A.s ∈ func.rules(B) then s ∈ ProvidesA
    - if B.s ∈ CallsA then (s,A) ∈ sec.rules(B)

Trickier Example

[redacted]

secure CHANGE

- **Applet ePurse:**

•Provides = {*transferMoney*}
•Calls = {}
•Sec.rules = {*transferMoney* → {*jTicket*}}
•Func.rules = {}

- **Applet jTicket:**

• Provides = {*ageDiscount, loyaltyDiscount*}
• Calls = {*ePurse.transfer_money*}
• Sec.rules ={ *ageDiscount* → {*IDapplet*}, *loyaltyDiscount* → {*ePurse*)}
• Func.rules = {*ePurse.transfer_money*}

- **Now we update ePurse**

• Provides = {*transferMoney*}
• Calls = { jTicket.ageDiscount }
• Sec.rules = {*transferMoney* → {*jTicket*}}
• Func.rules = {}

- **What happens?**

Secure Platform

- **A platform Θ remains secure during evolution**
  - **This is what you really want after each update**
  - **For every applet the traces of real executions respects its security and functional rules**
    - Whenever somebody calls you it is authorized
    - Whenever you need to call an essential service it is still there (provided it was there before)
- **Security and functionality in terms of Contracts**
  - **Contracts do not violate Global Policy**
  - **Claims are consistent with bytecode**
  - **Otherwise update is rejected**
- **Need to show the two coincide.**

Security Theorem

[redacted]

secure CHANGE

- **IF Platform was secure before the update,**
- **& IF shareable interfaces are only means for inter-app communication**
- **& IF Claim Checker and the Policy Checker are sound and accepted an update at the loading time,**
- **THEN evolved platform will be secure.**
  - **Proving by contradiction that if security or functionality is broken on the platform, then either the ClaimChecker, or the Policy Checker will reject the update**
- **Still an Engineering gap**
  - **In theory it could work for Application IDs in contracts,**
  - **in practice we may need to weaken the claim to Package Ids**
- **Depends on what we can implement in the claim checker**

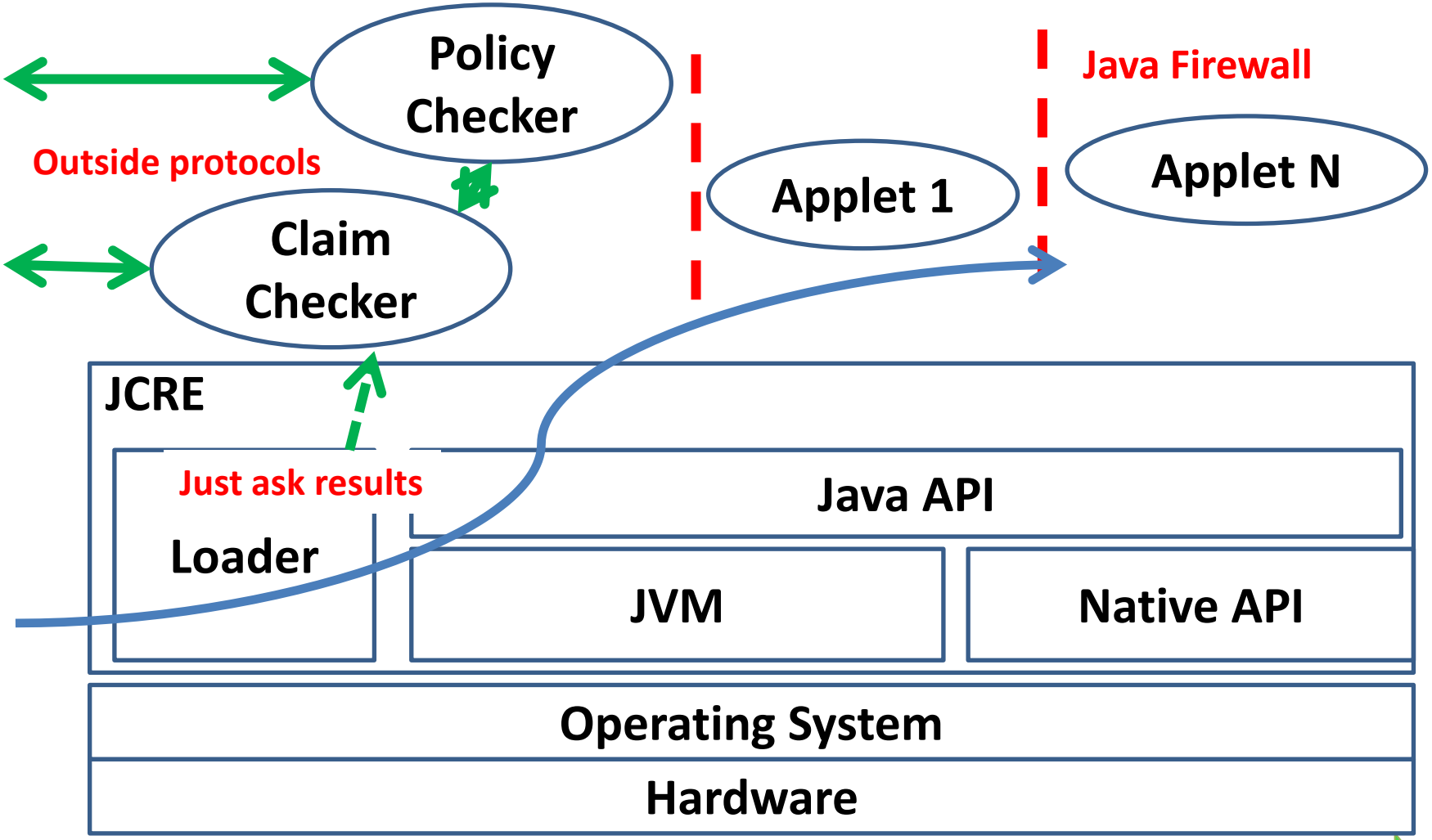
The talk plan

[redacted]

secure CHANGE

- **Where's Trento?**
- **The rara avis of multi-application smart-cards**
- **Security-by-Contract for smart cards**
- **A (thin) slice of theory**
- **A (larger) slice of engineering**
- **Open problems**

Our First Architecture



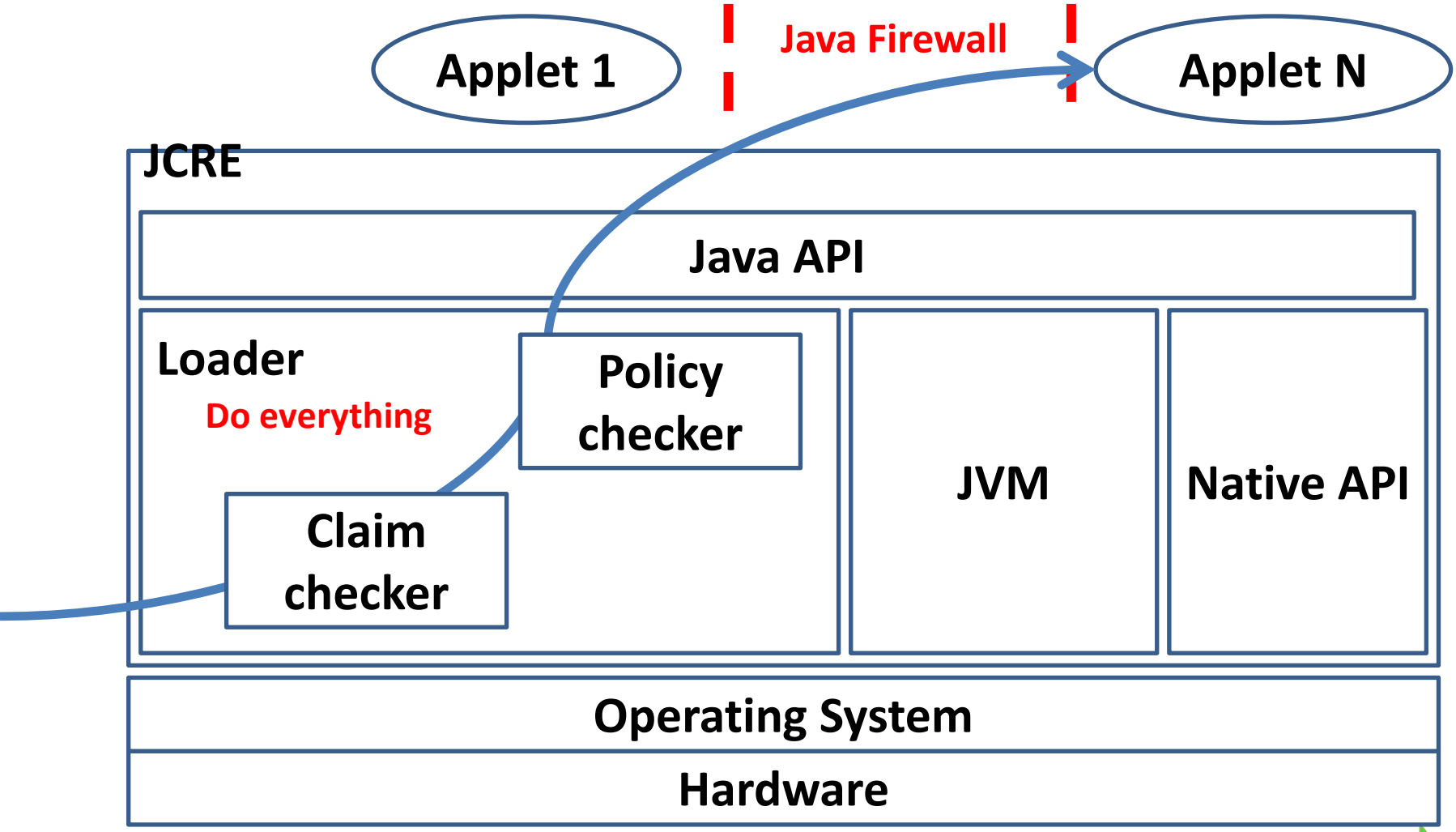
First Engineering problem

[redacted]

secure CHANGE

- **Implemented Policy Checker**
  - **POLICY'11 short paper**
  - **Footprint of checker 11KB and contracts 2KB**
- **Require changing existing update protocols**
  - **1stprotocol with policy checker**
  - **2nd protocol with claim checker**
  - **3rd protocol is standard loading plus check results of 1+2**
- **Loader can trust policy checker, what about claim checker?**
  - **Needs signatures and certification**
  - **Too small improvement to justify change update protocol**

Our Second Architecture



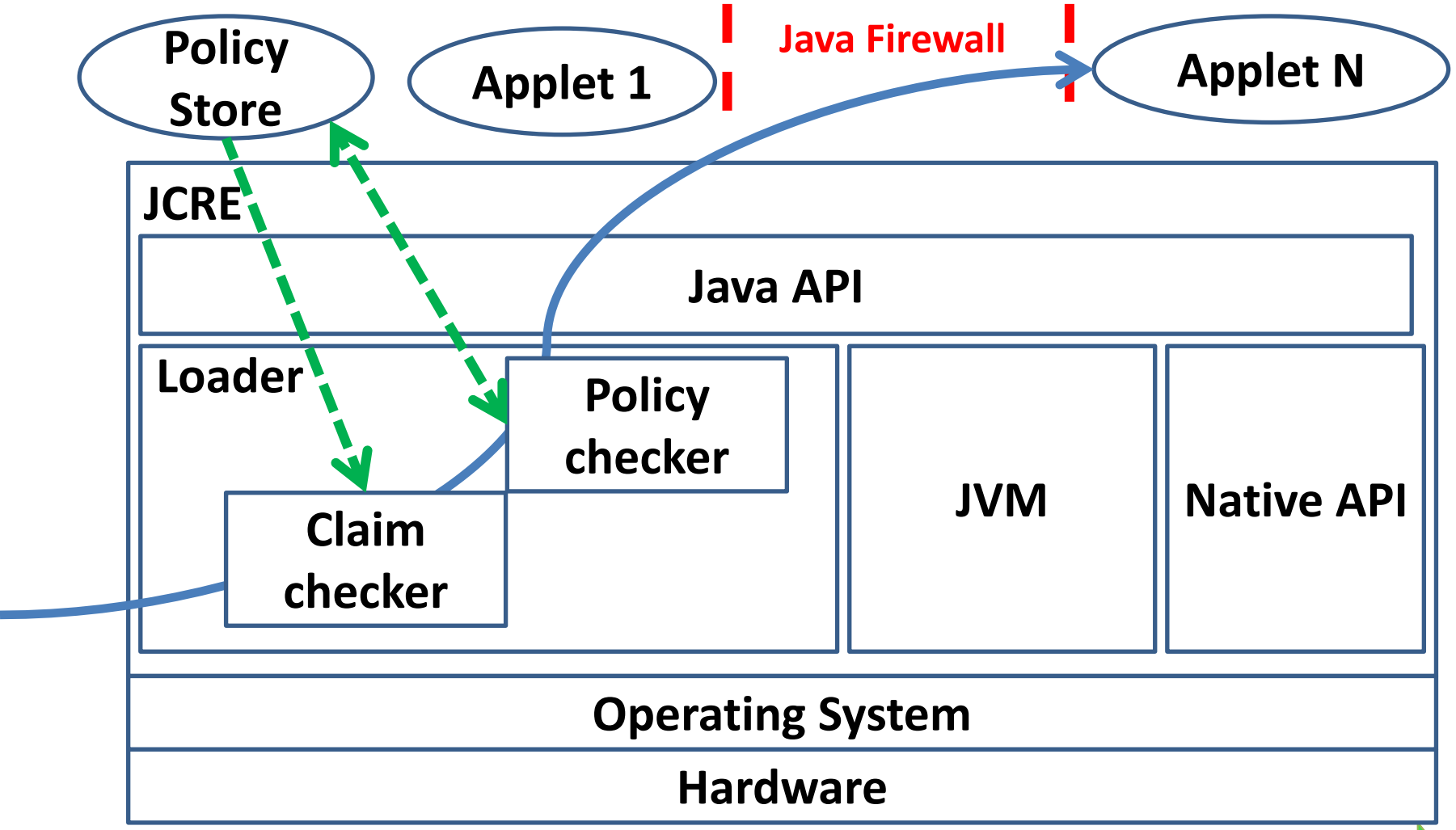
Second Engineering Problem

[redacted]

secure CHANGE

- **More Effective and Efficient**
  - **Loader no longer trust external checks of code**
  - **Eliminate checks of signatures beside standards**
  - **Both checkers can be implemented in C**
- **But where do we put the policy?**
  - **We need to retrieve it and store it somewhere…**
  - **but loader is NOT loaded in the EEPROM**
    - We could have a “static int policy[]’’ but that’s not going to work in the ROM

Our Third Architecture



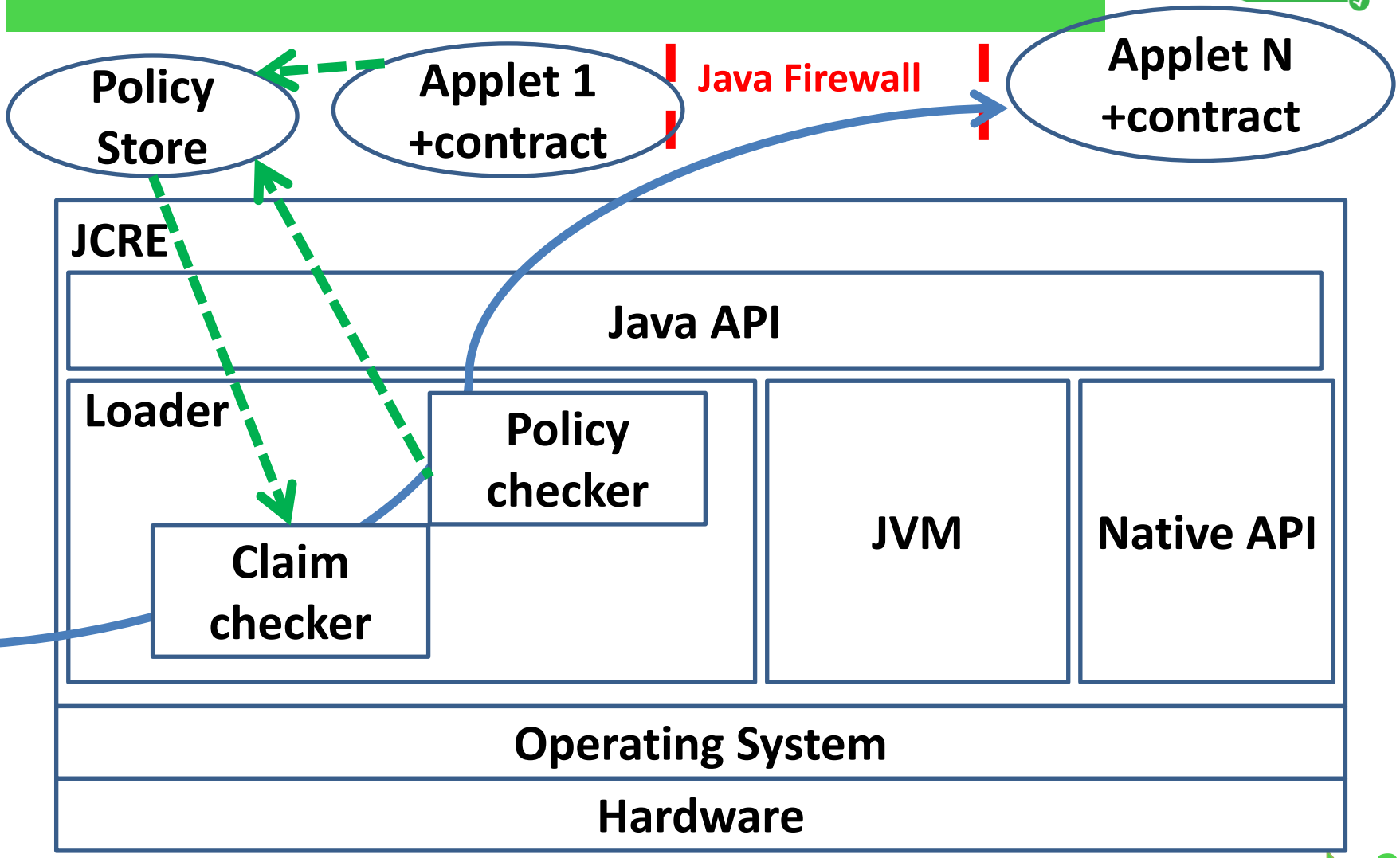
Third Engineering Problem

[redacted]

secure CHANGE

- **C and Java don't mix well**
  - **The loader can "easily" invoke the Policy Store applet at the beginning of the process andpass reference to it to the loader**
    - Just need a Java shell onto the loader
  - **but how to tell it the result at the end??**
    - It must be the checked contract and nothing else
- **Who's giving the contract to the checker?**
  - **Must change the protocol of update…**

Our Third Architecture



Engineering Idea

[redacted]

secure CHANGE

- **Each Applet includes contract in java package**
  - **No need to send it separately**
  - **Arrives and leaves with applet**
  - **Neutral: contract update requires re-running claim checker**
  - **Cons: contract update requires code update**
    - But in this way claim checker re-run is automatic!
- **Policy store references applet contract**
  - **Keep efficiency of C implementation with Java flexibility**
- **Checkers do not need trust anyone**
- **Next validation by Smart card manufacturer**

The talk plan

[redacted]

secure CHANGE

- Where's Trento?
- The rara avis of multi-application smart-cards
- Security-by-Contract for smart cards
- A (thin) slice of theory
- A (larger) slice of engineering
- **Open problems**

Trickier Example

[redacted]

secure CHANGE

- **Applet ePurse:**

•Provides = {*transferMoney*}
•Calls = {}
•Sec.rules = {*transferMoney* → {*jTicket*}}
•Func.rules = {}

- **Applet jTicket:**

• Provides = {*ageDiscount, loyaltyDiscount*}
• Calls = {*ePurse.transferMoney*}
• Sec.rules ={ *ageDiscount* → {*IDapplet*}, *loyaltyDiscount* → {*ePurse*)}
• Func.rules = {*ePurse.transfer_money*}

- **We update ePurse**
- **Update is accepted**
- **What happens later if jTickets wants to drop access to ePurse?**

• Provides = {*transferMoney*}
• Calls = {jTicket.loyaltyDiscount}
• Sec.rules = {*transferMoney* → {*jTicket*}}
• Func.rules = {jTicket.loyaltyDiscount}

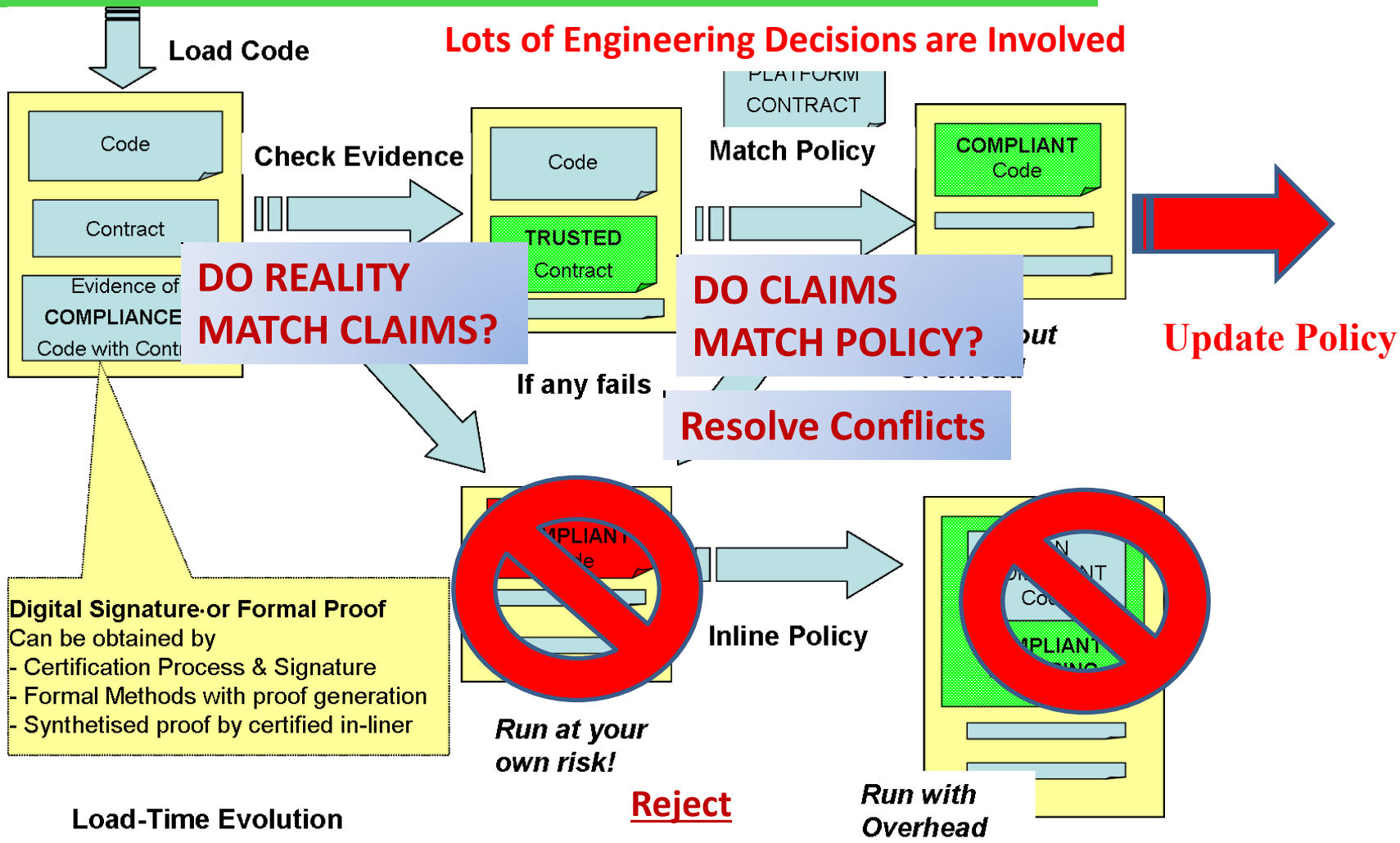
A Conflict Resolution Componnet?

[redacted]

secure CHANGE

- **What happens if ePurse owner wants it to be removed from the platform?**
  - **jTicket needs the service ePurse.transfer_money**
  - **But ePurse doesn't want (now) to give him this**
- **Two possibilities:**
  - **to forbid ePurse to be removed OR**
  - **to remove ePurse and make jTicket unselectable.**
- **(Automatic) Conflict resolution requires investigation of stakeholders (security domains) hierarchy.**

Conclusions: SxC for Smart-Cards



[redacted]

secure CHANGE

Send us your applets!

fabio.massacci@unitn.it
gadyatskaya@dit.unitn.it