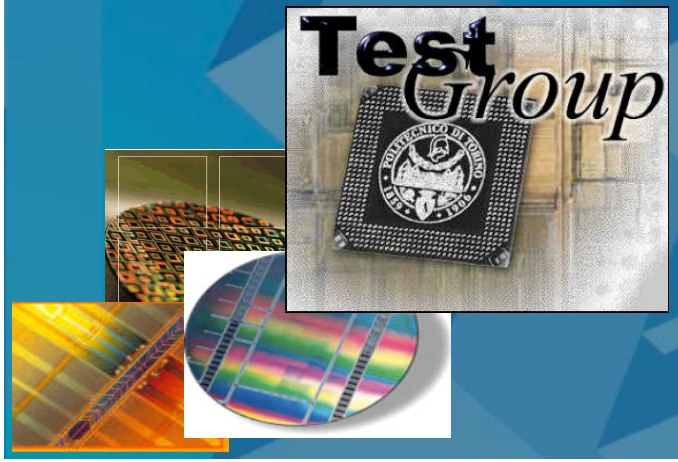
20-21 March 2007 GEMALTO (Paris)

Increasing the integration between the un-trusted application and the monitor

S. Di Carlo, D. D'Aprile

TESTGROUTP - Politecnico di Torino (Italy)

www.testgroup.polito.it



Purpose

- To enhance the robustness of the trust model in RE-TRUST by increasing the integration between the monitor and the monitored application

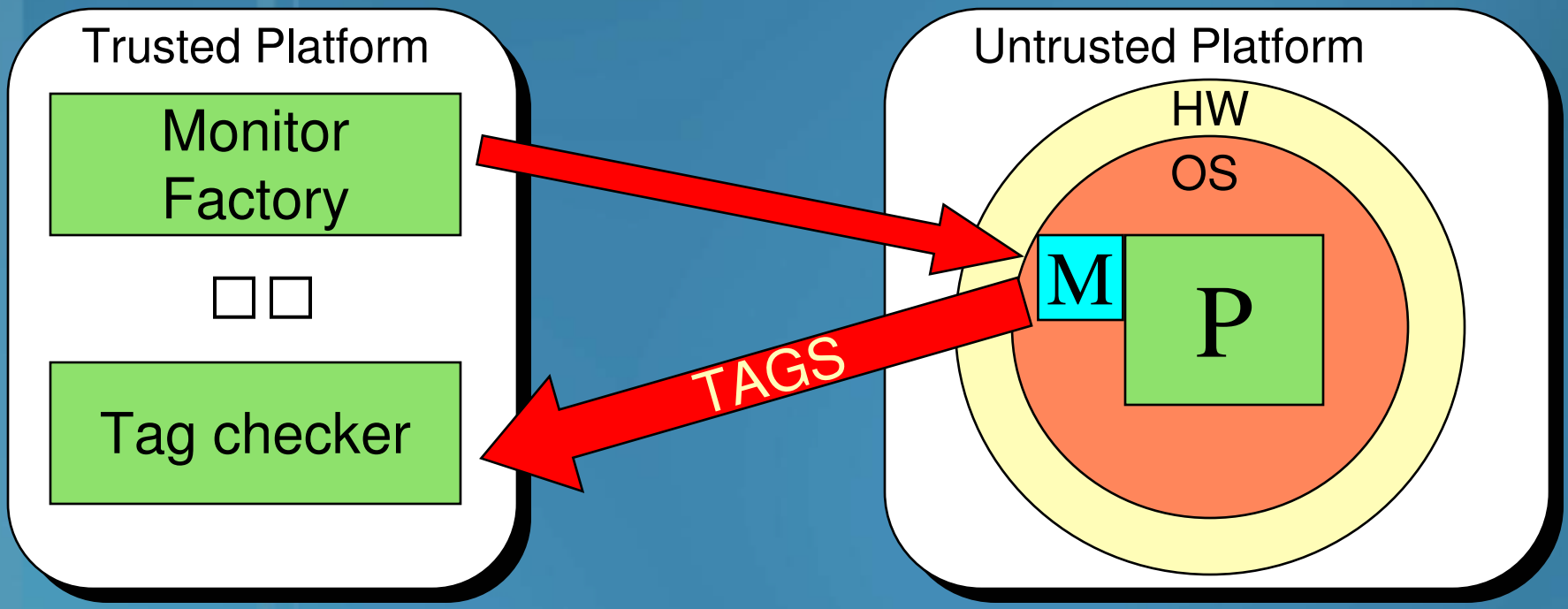
Outline

- Motivations
- Distributed monitor design
- Modified trust model
- Conclusions and future works

Outline

- Motivations
- Distributed monitor design
- Modified trust model
- Conclusions and future works

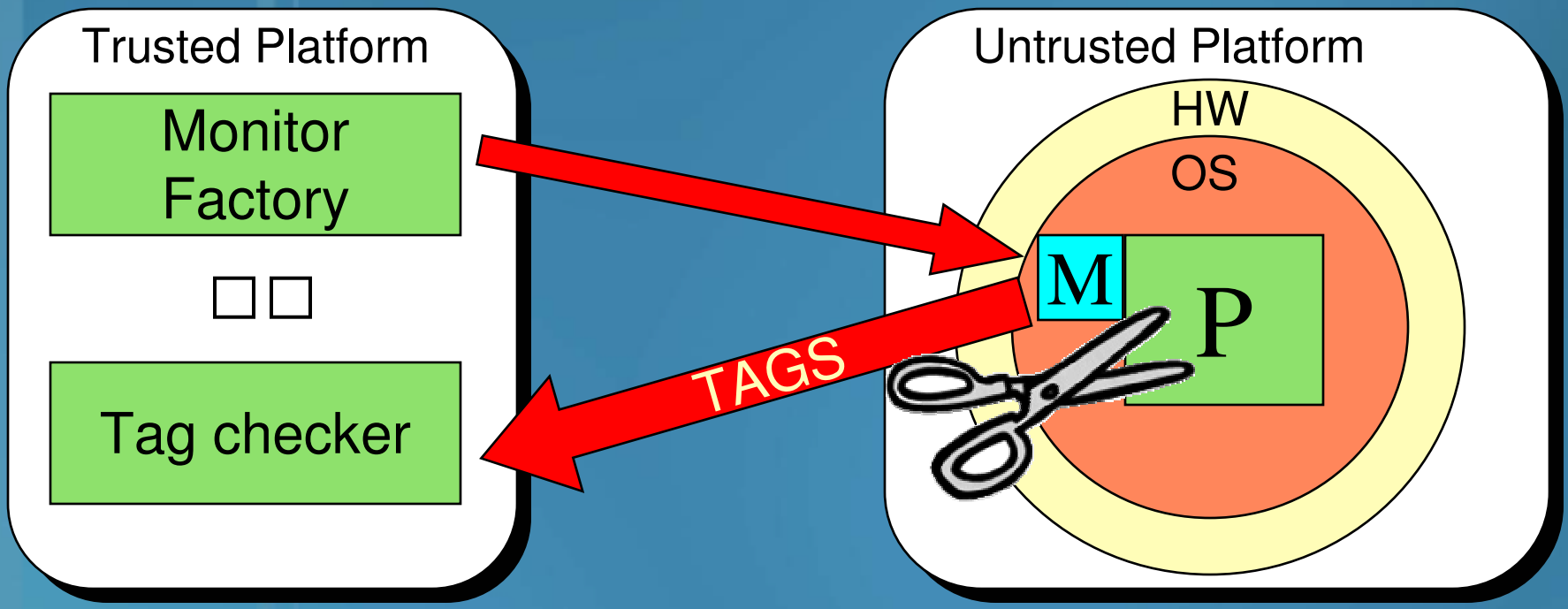
Motivations



Motivations

- Trust model key elements:
  - Application authenticity verification
    - Text and data segments
    - Libraries
    - Execution environment
    - Assertions
  - Authenticity tags flow
  - Replacement
  - Code obfuscation

Motivations



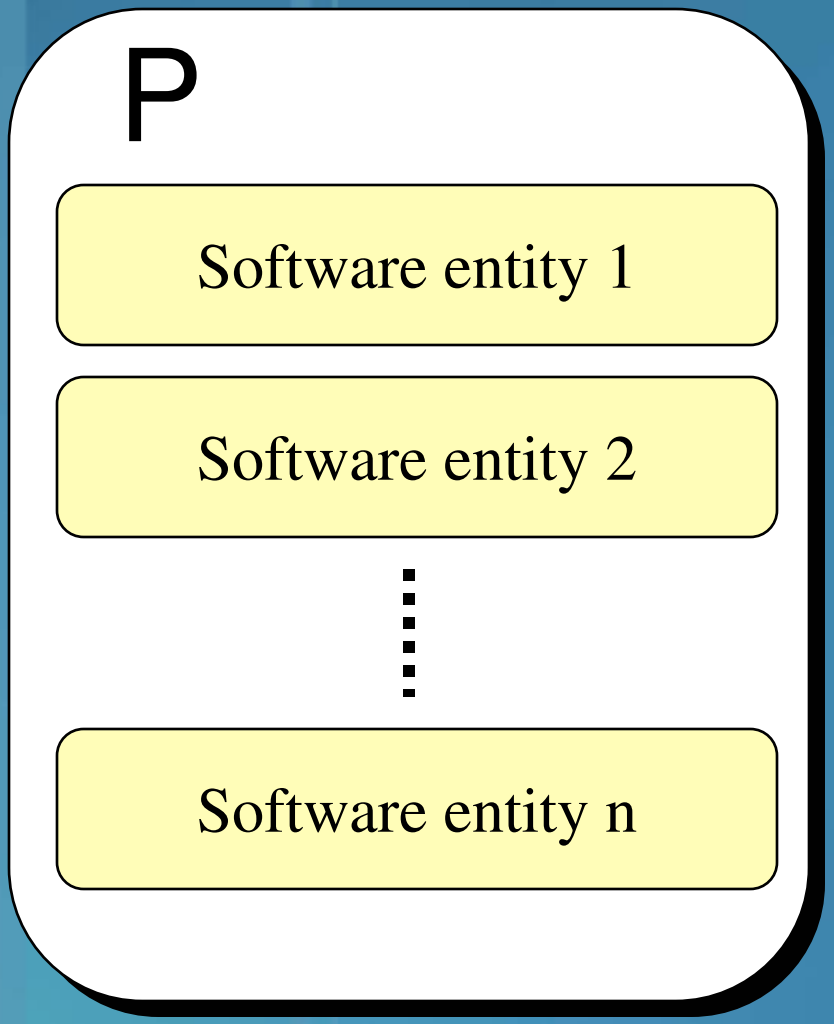
Motivations

- Monolithic monitors, loosely attached to the program can be easily disengaged
  - AOP based monitors
  - JVMTI based monitors
  - .NET based monitors
- The integration between M and P must be strong

Outline

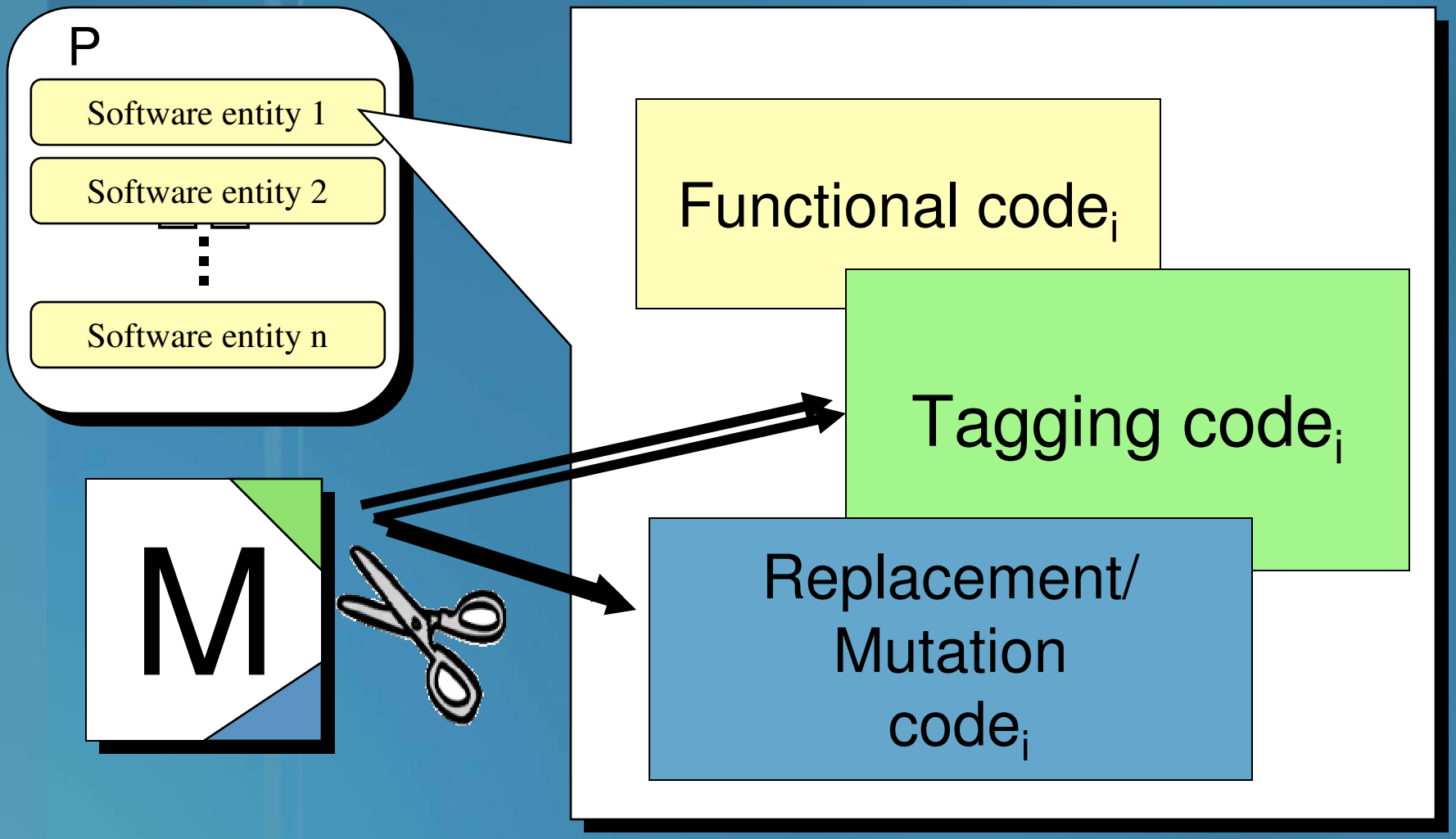
- Motivations
- Distributed monitor design
- Modified trust model
- Conclusions

Distributed monitor design



- Software entity as an abstraction of:
  - Functions
  - Classes
  - Modules
  - Library
  - ….

Distributed monitor design



Distributed monitor design

- **Functional code**: performs the functionalities the entity is designed for
- **Tagging code:**
  - Monitoring of static and dynamic parameters of P
  - Generation of the secure tags flow
- **Replacement/Mutation Code**:
  - **Replacement**: manages the replacement of the tagging code
  - **Mutation**: a mutation function dynamically modifies the entity to get a new version with different tagging code

Distributed monitor design

- Advantages:
  - M is spread in P
  - Each en[illegible]ay exploit d[illegible]n diver[illegible]y to implement [illegible]rent mon[illegible]ng mech[illegible]m
- Is this ap[illegible]roac[illegible]ugh[illegible] to gua[illegible]tee the required level o[illegible]ust?

NO

Distributed monitor design

- Additional requirements
  - Tagging code and functional code interleaved
    - Software guards? [H. Chang and M. Atallah. Protecting software code by guards. In Proceedings of ACM Workshop on Security and Privacy in Digital Right Management, 2002]
    - Tagging code replacement not trivial:
      - Replace the full entity?
      - Use of mutation

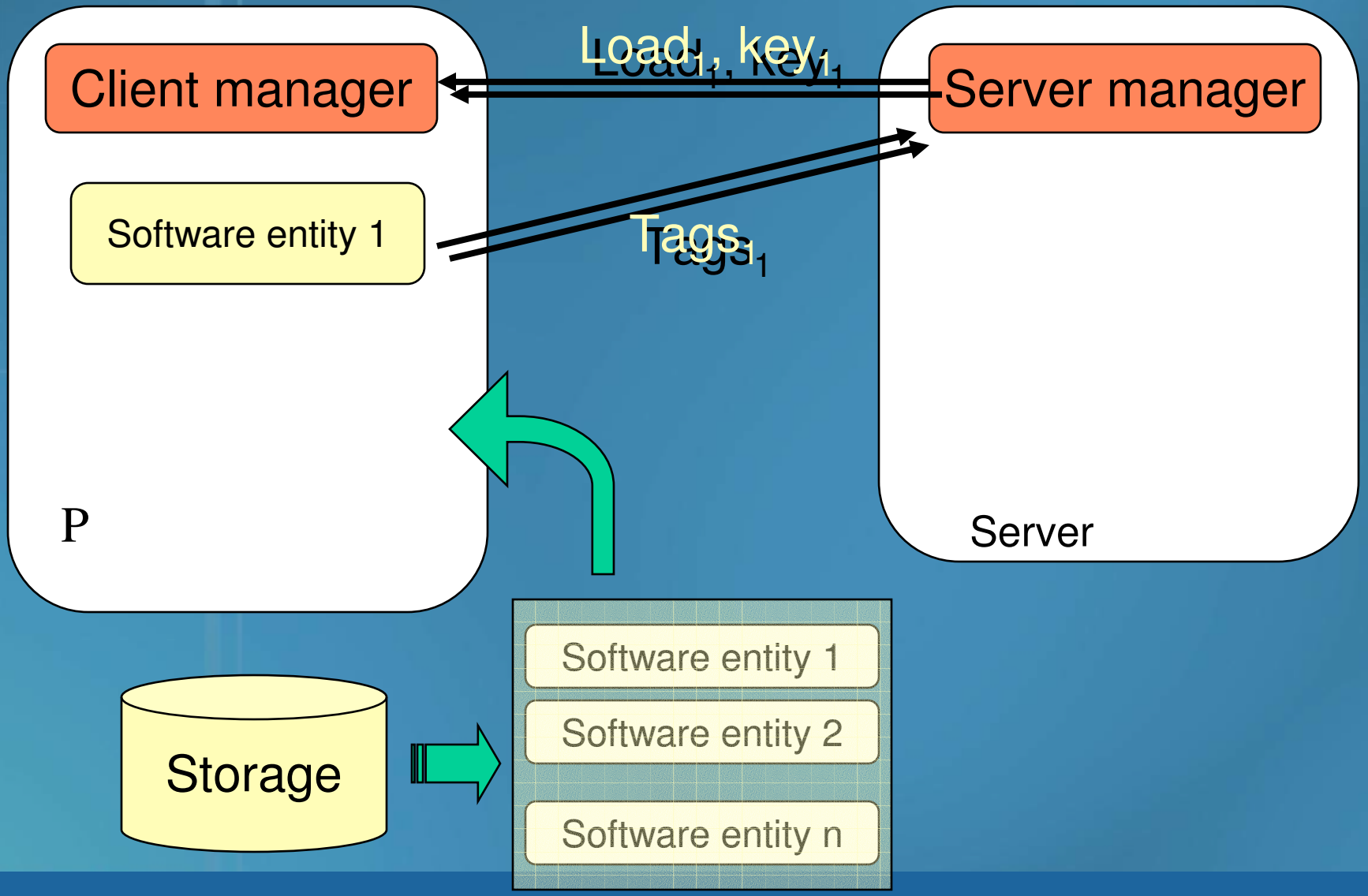
Distributed monitor design

- Code deployed in an encrypted form?
  - Entities decrypted only when executed
  - Once the entity is loaded into the main memory the attacker has the potential opportunity of analyzing the clear code
    - We have to reduce this opportunity

Outline

- Motivations
- Distributed monitor design
- Modified trust model
- Conclusions ad future works

Modified trust model



Modified trust model

- Server manager commands:
  - Load: it asks the client manager to load a new entity and it provides the decryption key
  - Unload: it asks the client manager to garbage a software entity when not needed:
    - Reduces the time an entity is in main memory (clear code)
    - Other entities can check that this mechanism is no tampered with

Modified trust model

- Replacement: it communicates with the replacement code of an entity to:
  - Send a new version of the entity (encrypted with a new key) to be stored for the next execution
  - Send a mutation command to the entity

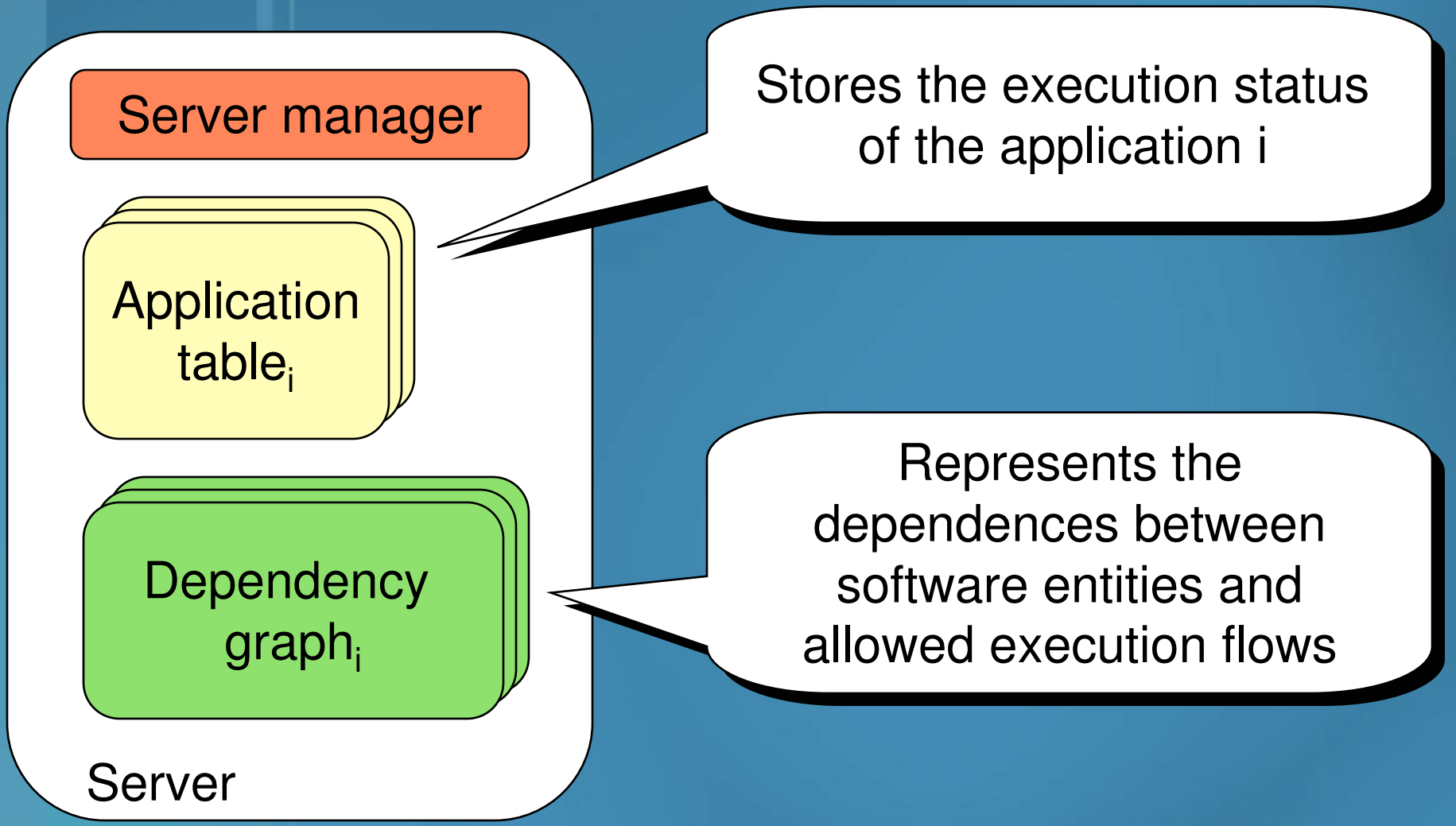
Modified trust model

- The client manager
  - Reacts to the command received by the server manager
  - Loads, decrypts and starts the execution of software entities

Modified trust model

- Requirements:
  - The server has to know the execution status of the application
  - The server has to know the dependencies (control flow) between software entities

Modified trust model



Modified trust model

- Dependency graph
  - Obtained at compile time
  - Trade-off between granularity and complexity
  - Used to schedule the software entities execution
  - Used to perform a remote (server side) control flow integrity checking

Outline

- Motivations
- Distributed monitor design
- Modified trust model
- Conclusions and future works

Conclusions and future works

- We presented a distributed architecture for the RE-TRUST monitor and an extension of the trust model that allows:
  - Strong integration between monitor and monitored application
  - Increased effort to reverse engineering the application
  - Distribution of the checking activities between application and trust server

Conclusions and future works

- Future works:
  - Formalization of the idea
  - Investigation on mutation mechanisms
  - Applicability analysis on different HW/SW platforms:
    - PowerPC, Intel, …
    - Native code, Java, .NET
  - Prototype platform:
    - VoIP system is the candidate platform