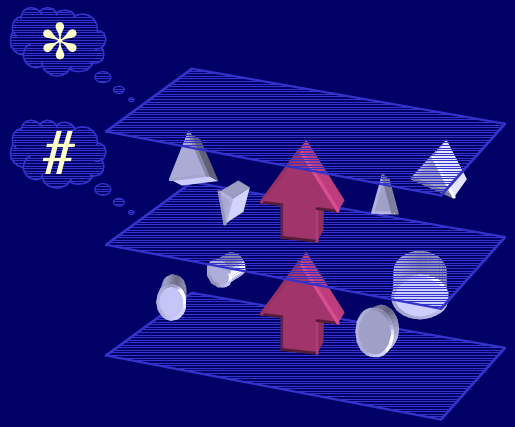


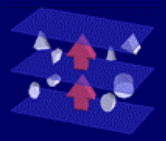
Principles of Multi-Level Reflection for Fault Tolerant Architectures

PRDC'02 (Dec 16-18 2002, Tsukuba, Japan)

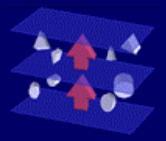
François Taïani, Jean-Charles Farbre, Marc-Olivier Killijian

CNRS LAAS

Context



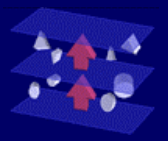
- Software COTS now in systems with high FT requirements
- Market products don't meet dependability requirements
  ⇒ Adaptation needed
- Dependability is a cross-cutting concern
  ⇒ **Reflective architectures** seem a good choice
- But multi-component systems are a challenge for reflection



Outline

- Introducing reflection for fault-tolerance
- The Fault-Tolerance of Multi-component systems
- Our proposal: Multi-Level Reflection
- Conclusion

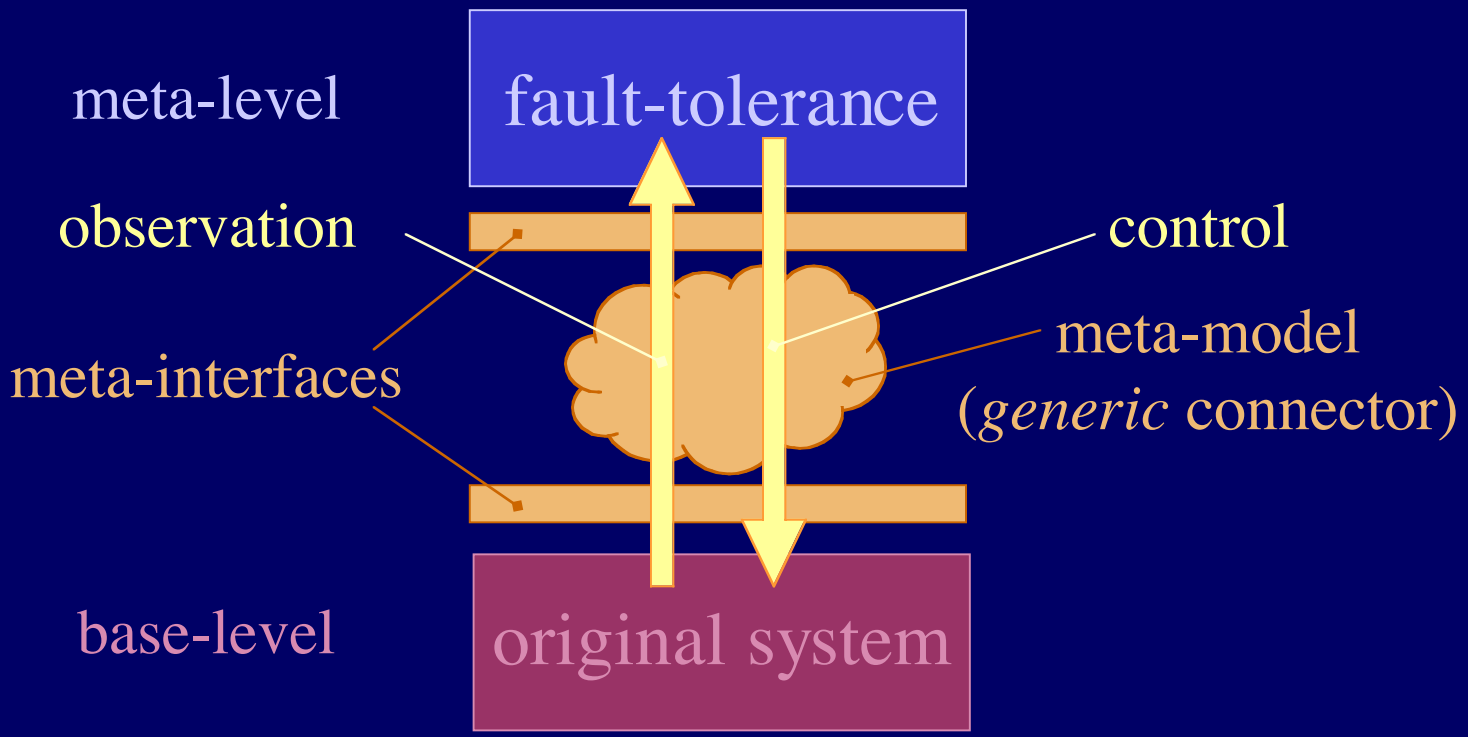
Outline



- Introducing reflection for fault-tolerance
- The Fault-Tolerance of Multi-component systems
- Our proposal: Multi-Level Reflection
- Conclusion

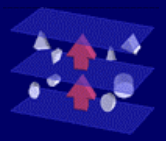
What is Reflection?

"the ability of a system to think and act about itself"



☞ separating fault-tolerance from functional concerns

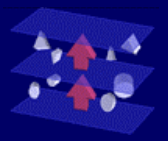
Reflection & Fault-Tolerance



- Meta-model provides observation and control features that are needed to implement fault-tolerance
  - State capture (observation) / State recovery (control)
  - Method interception (observation) / Duplication (control)
  - Non-deterministic decision points
  - …

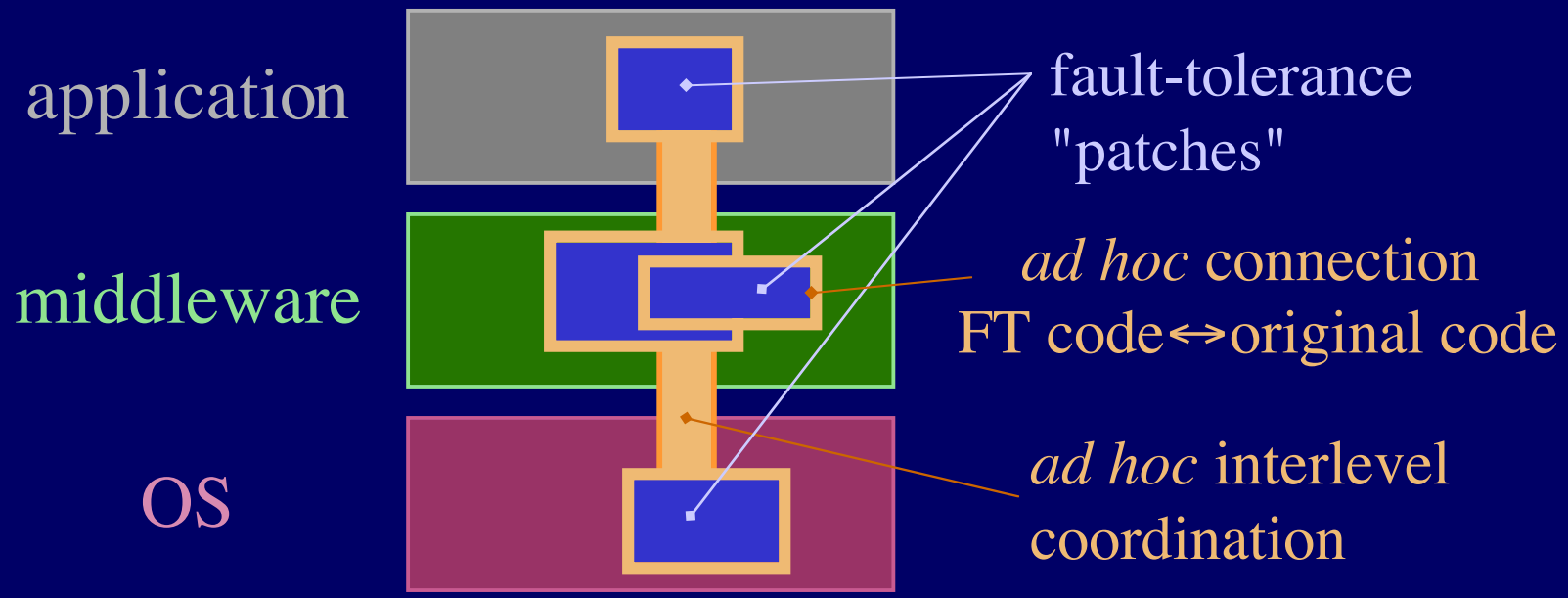
- In a multi-component system:
  - Information/controls possible in different layers / abstraction levels
  - Higher levels (application, language): partial info / rich semantics
  - Lower layers (OS, middleware): complete info / poor semantics

Outline

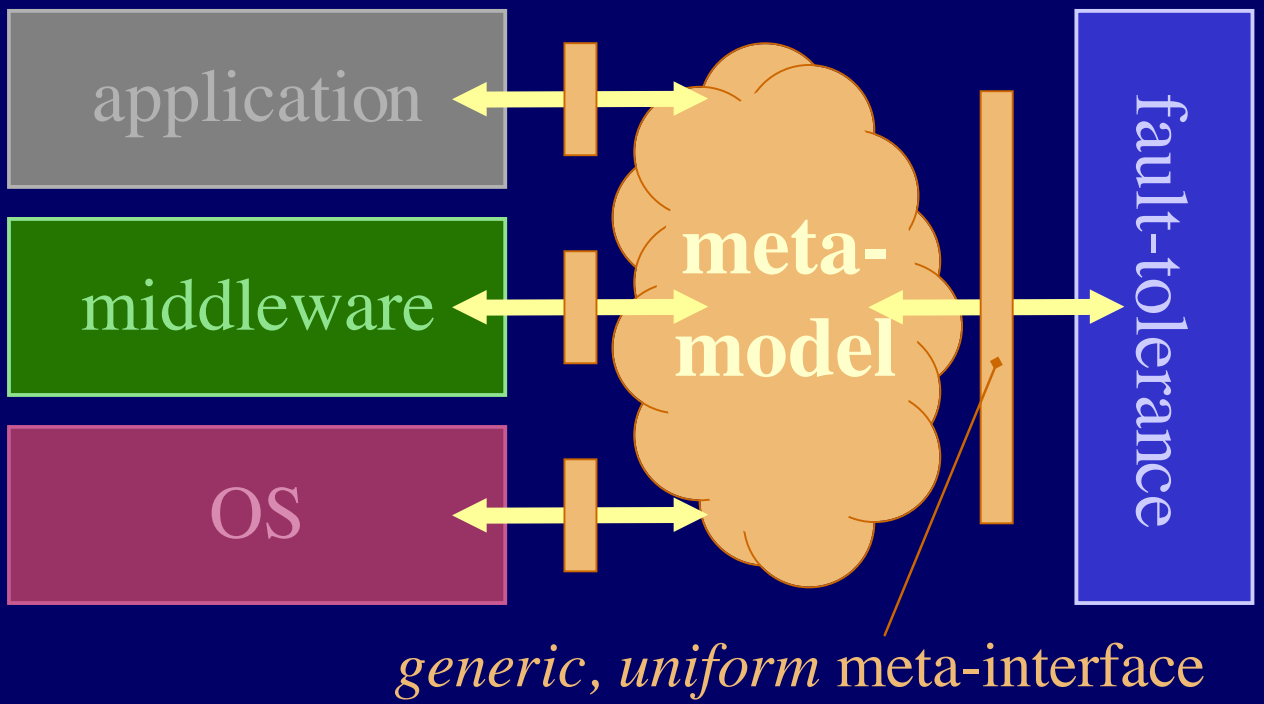


- Introducing reflection for fault-tolerance
- The Fault-Tolerance of Multi-component systems
- Our proposal: Multi-Level Reflection
- Conclusion

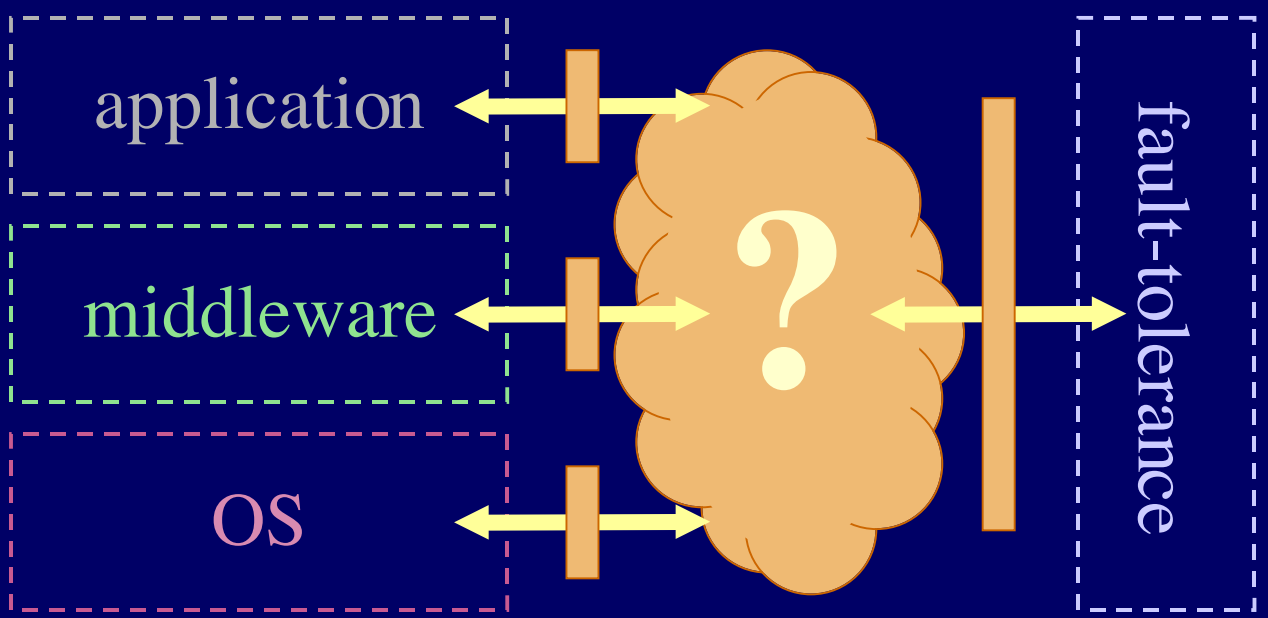
Ad Hoc FT in Multi-Level Systems



Reflective Approach

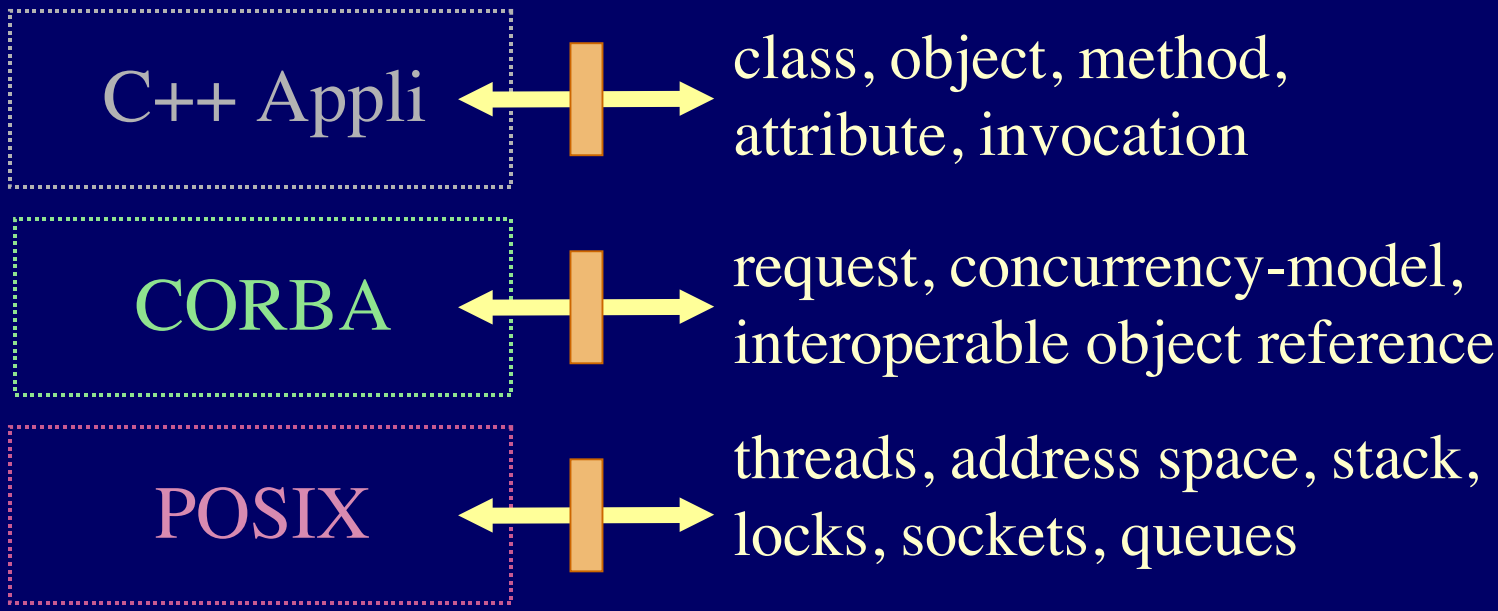


FT in Multi-Level Systems



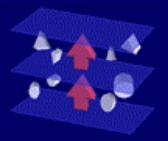
- What information/control from which level?
- How to aggregate information/control from ≠ levels?

Example



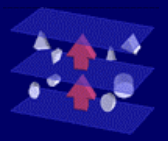
- What information/control from which level?
- How to aggregate information/control from ≠ levels?

Outline



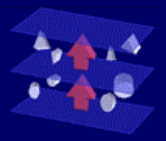
- Introducing reflection for fault-tolerance
- The Fault-Tolerance of Multi-component systems
- Our proposal: Multi-Level Reflection
- Conclusion

Multi-Level Reflection



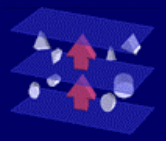
- 1. Construct a meta-model for each level / layer
- 2. Analyze inter-level dependencies & coupling
- 3. Aggregate single meta-models into a system wide model
- 4. Use system wide meta-model for fault-tolerance

Multi-Level Reflection



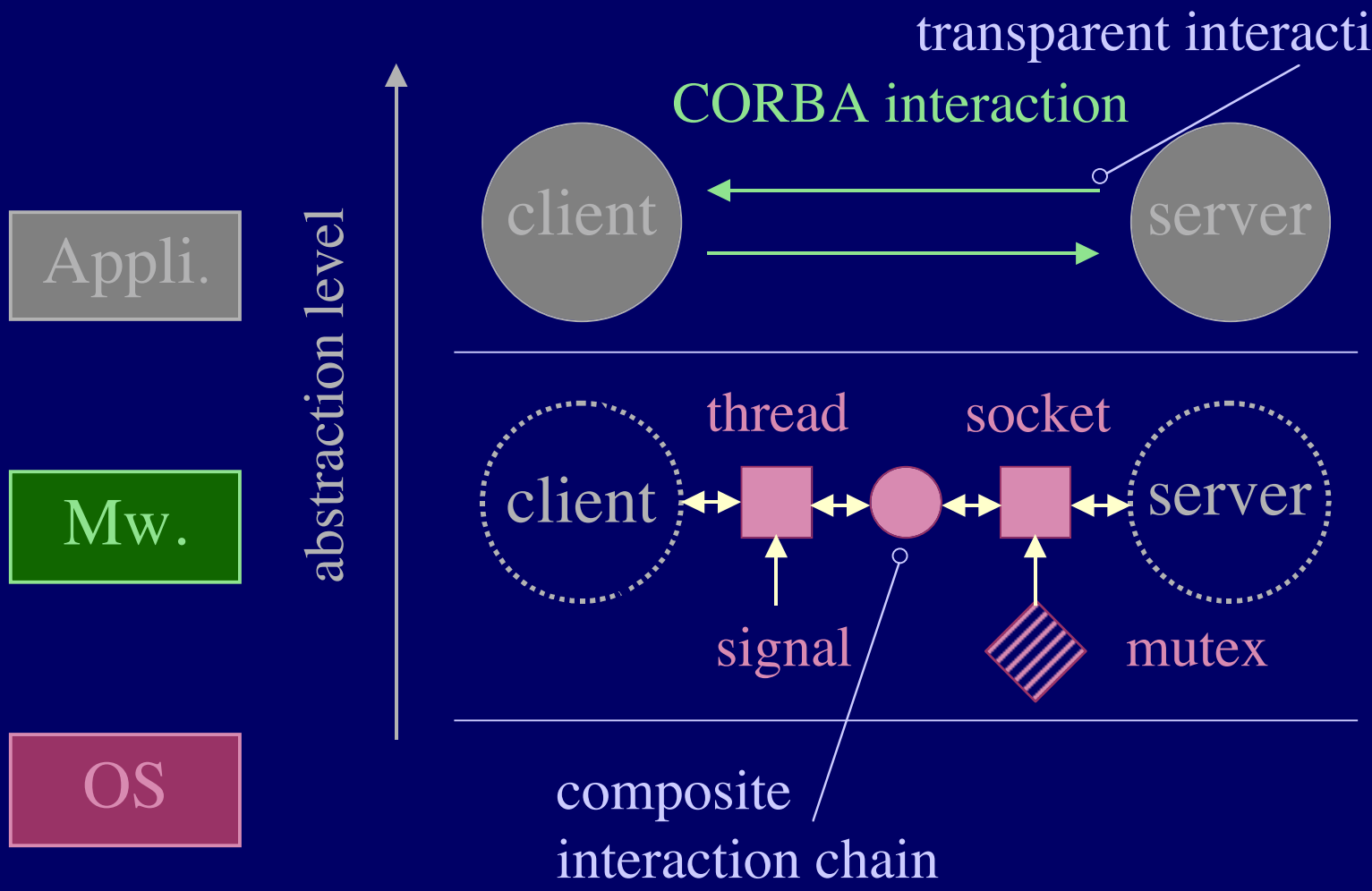
- 1. Construct a meta-model for each level / layer
- 2. Analyze inter-level dependencies & coupling
- 3. Aggregate single meta-models into a system wide model
- 4. Use system wide meta-model for fault-tolerance

Inter-Level Coupling (I)



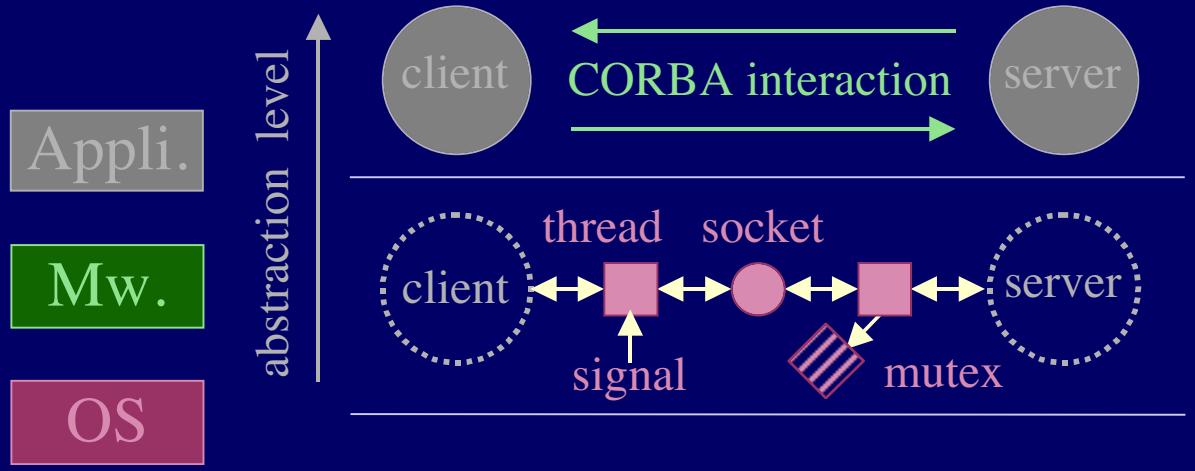
- A Level = 1..n COTS = A set of interfaces =
  - Concepts
  - Primitives / base entities (keywords, syscalls, data types, …)
  - Rules on how to use them
- (concepts, base entities, rules) = programming model
  - Very broad notion (includes programming languages)
  - Self contained
- Base entities "a-tomic" within that programming model
  - Can't be split in smaller entities within the programming model.
  - Implemented by more elementary entities within the component.
  - Implementation is internal ⇒ hidden to component user.

Inter-Level Coupling (II)



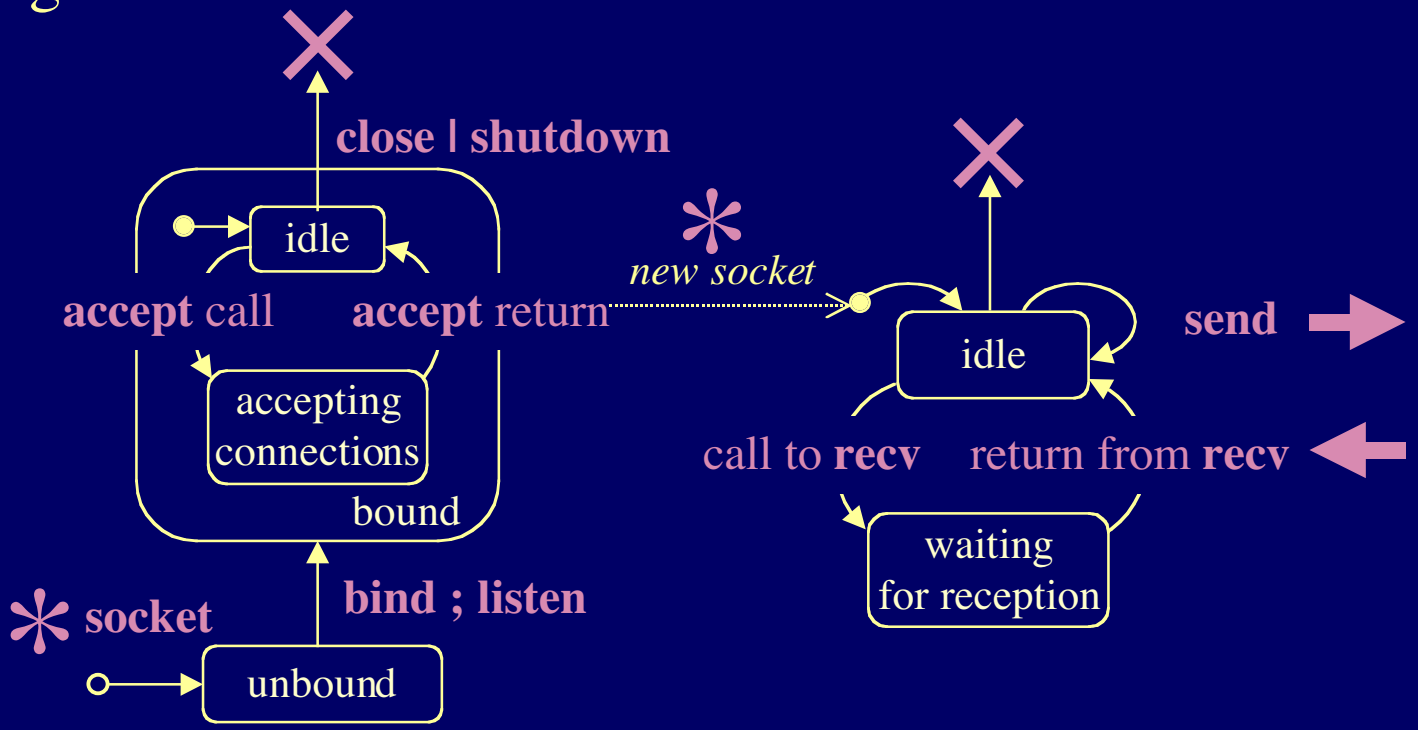
Inter-Level Coupling (III)

- Within a COTS :
  - Coupling between | emerging entities of next upper level
    and | implementation entities of lower levels
- Structural coupling relationships (“abstraction mappings”)
  - translation / aggregation / multiplexing / hiding
- Dynamic coupling relationships (“interactions”)
  - creation / binding / destruction / observation / modification

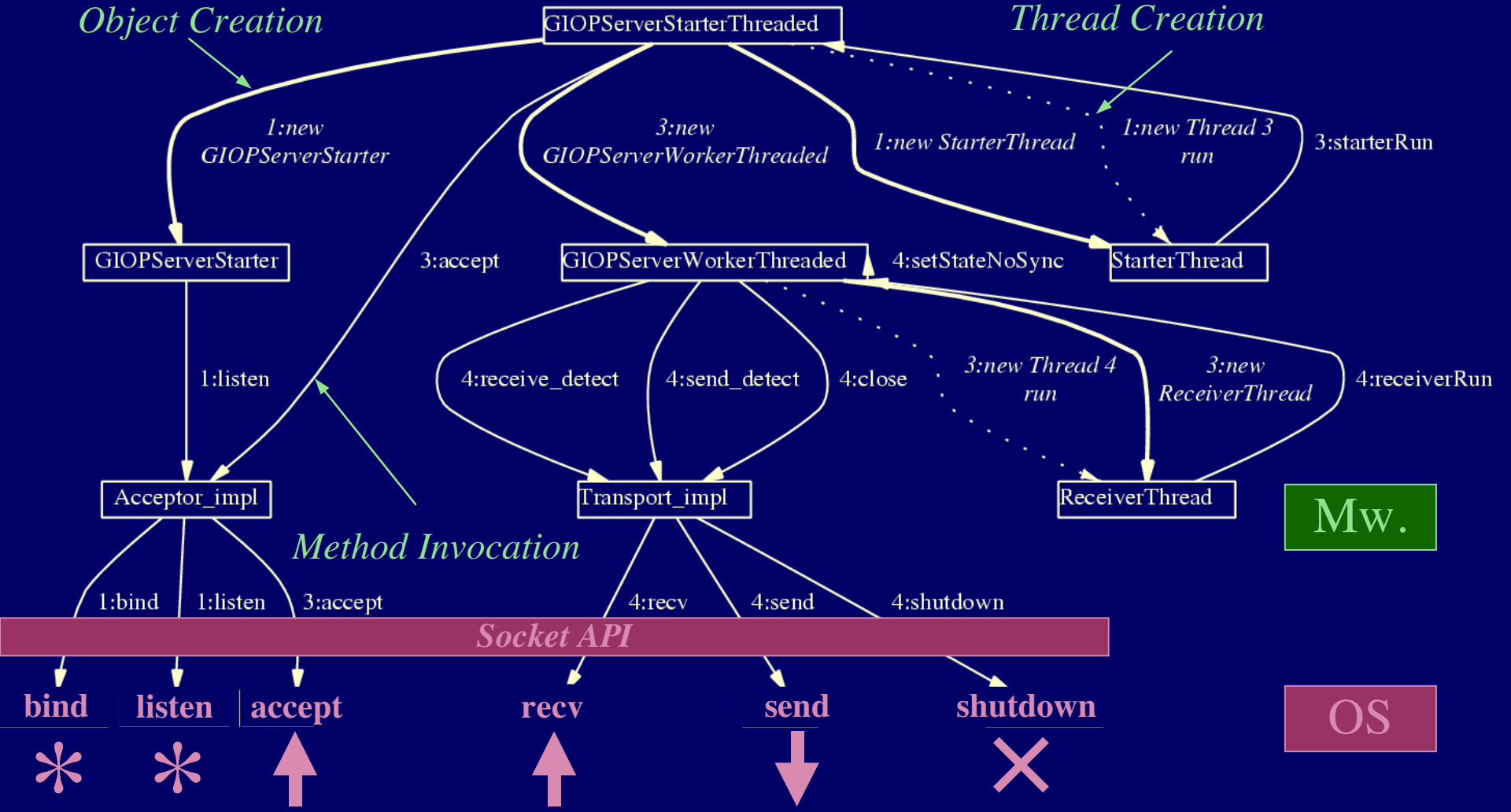


Example: Coupling POSIX / CORBA (I)

- Behavioral model of connection oriented Berkeley sockets as seen by the middleware programmer

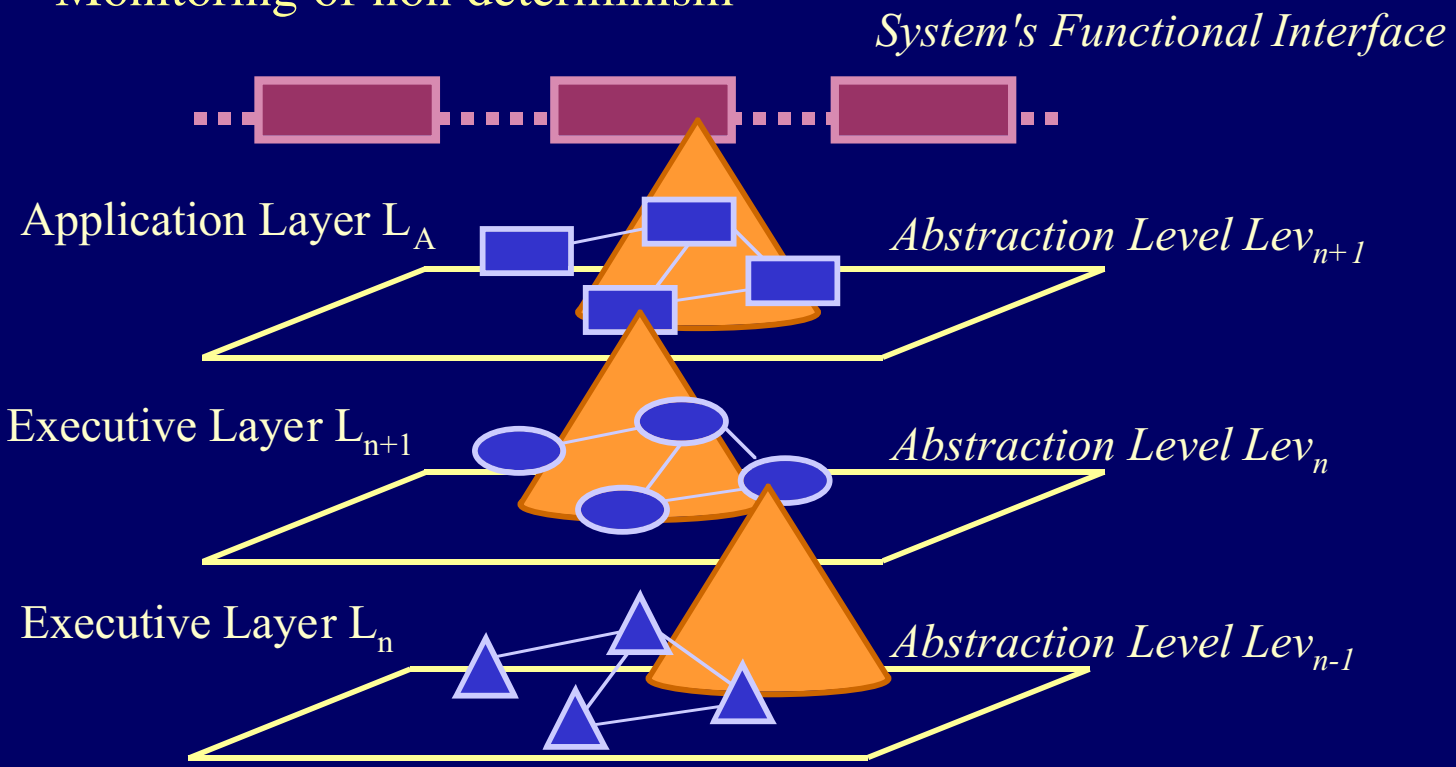


Example: Coupling POSIX / CORBA (II)



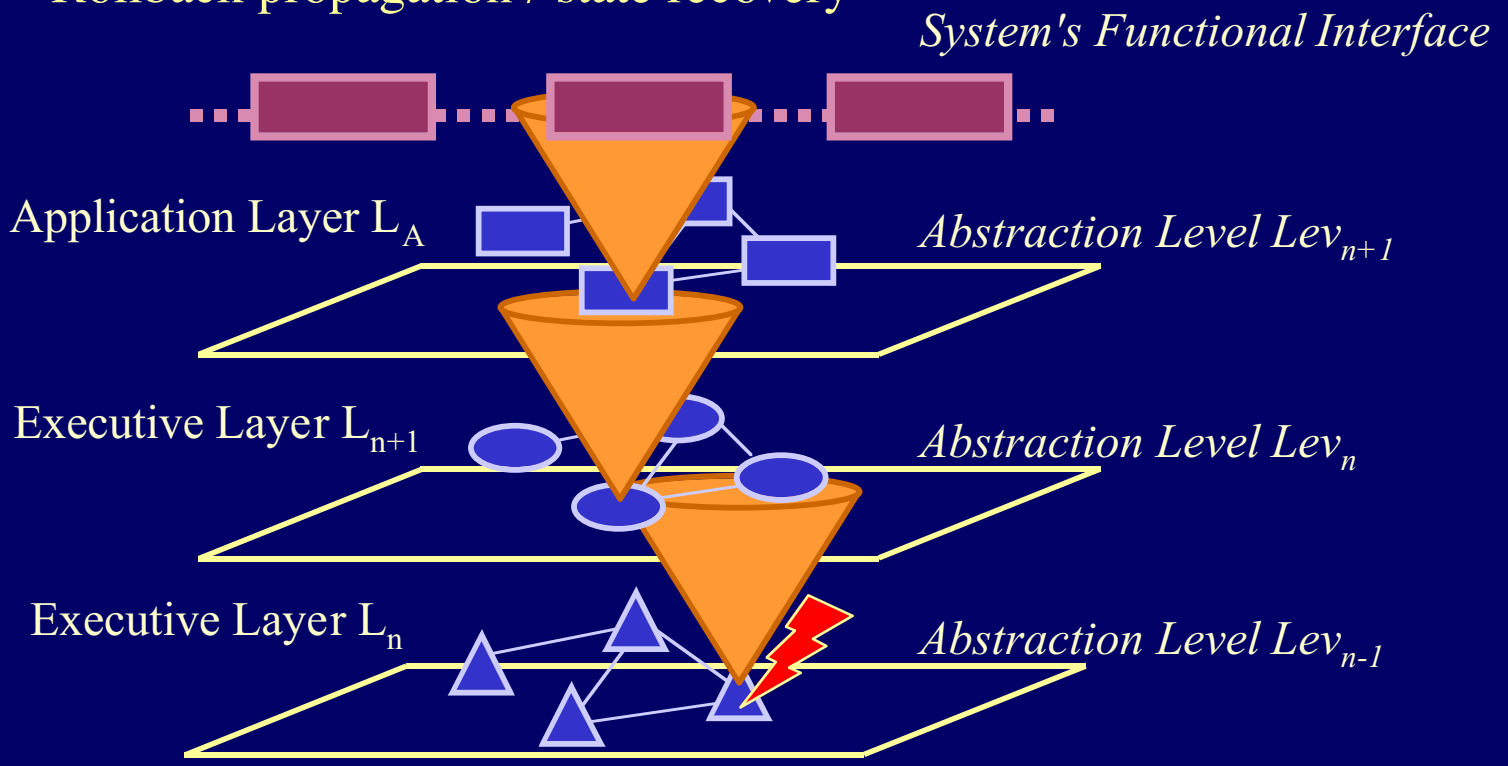
Using Multi-Level Reflection for FT (I)

- Top-down observation & control
  - State capture
  - Monitoring of non-determinism

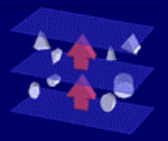


Using Multi-Level Reflection for FT (II)

- Bottom-up observation & control
  - Fault propagation analysis / confinement
  - Rollback propagation / state recovery

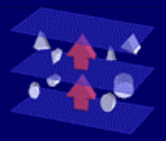


Outline



- Introducing reflection for fault-tolerance
- The Fault-Tolerance of Multi-component systems
- Our proposal: Multi-Level Reflection
- Conclusion

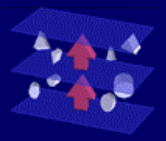
Conclusion



- Multi-Level Reflection
  (≈ Translucent Interfaces) can be very powerful
  - Accuracy of action & observation from lower levels
  - Power of correlation and understanding from higher level
- In practice:
  - Some low implementation decisions are equivalent when observed at higher levels (a.b ⇔ b.a, for instance memory management)
  - Identifying higher level patterns (for instance queue manag^{mnt} for Corba requests) can help reduce instrumentation costs.
- Join to be done between accuracy and understanding

⇒ "Adding higher level semantics to low-level entities"

Future Actions



- Finalize understanding of several ORBs + metamodel
- Start prototype implementation of multi-level meta-interfaces
- Proof of concept and evaluation with existing FT algorithms
- Adaptive Reflection (Customizable meta-models…)