A framework for managing dynamic service-oriented component architectures

Walter Rudametkin[1,2], Lionel Touseau[1], Didier Donsez[1], Francois Exertier[2]

1 LIG - ADELE
Grenoble University
{firstname}.{name}@imag.fr

2 BULL SAS
Echirolles, France
{firstname}.{name}@bull.net

Context

Building complex software systems

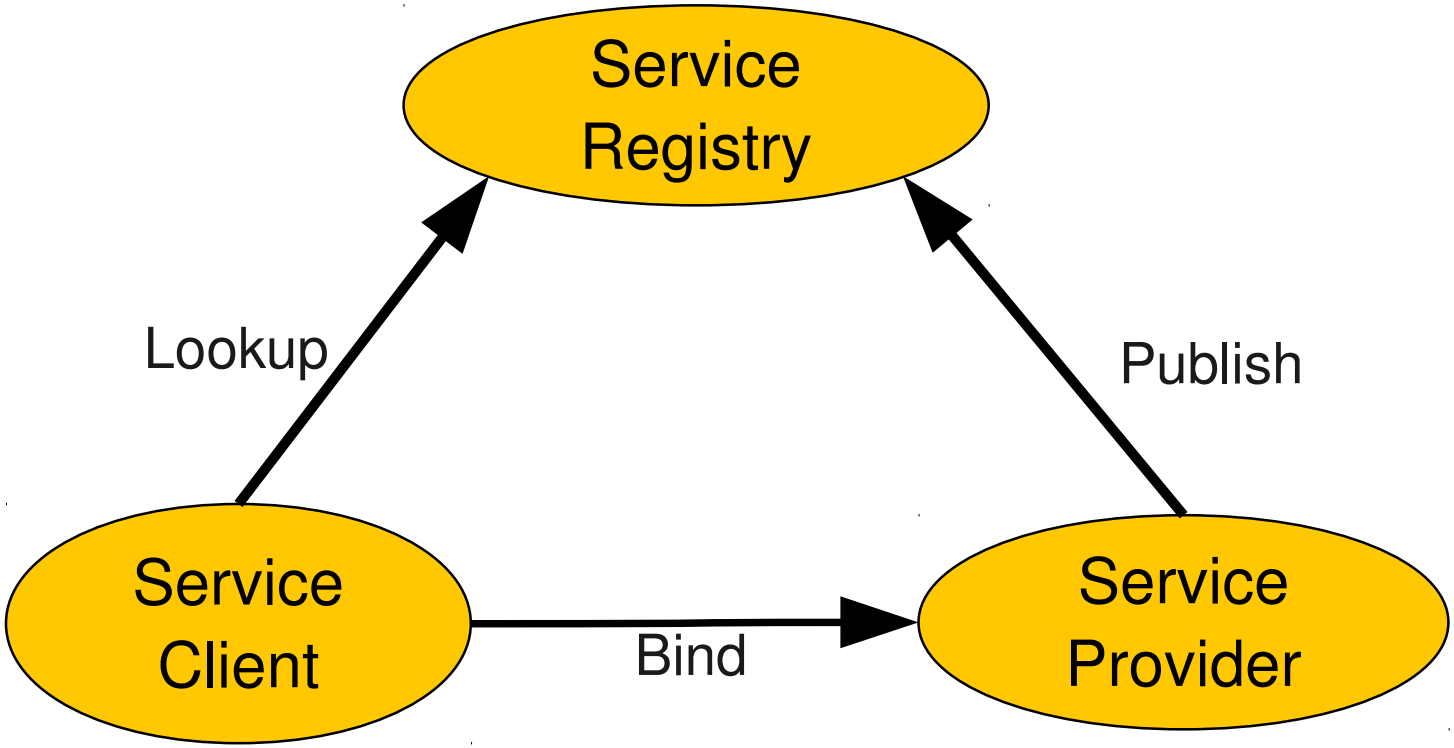
- Large systems
  - Millions of lines of code (eg.: Eclipse ~33 mloc)
- Entangled dependencies
  - Hundreds of different modules that must co-exist
- Run-time adaptation
  - We want the software to change/update @ runtime
  - Requires managing the architecture

Architecture management

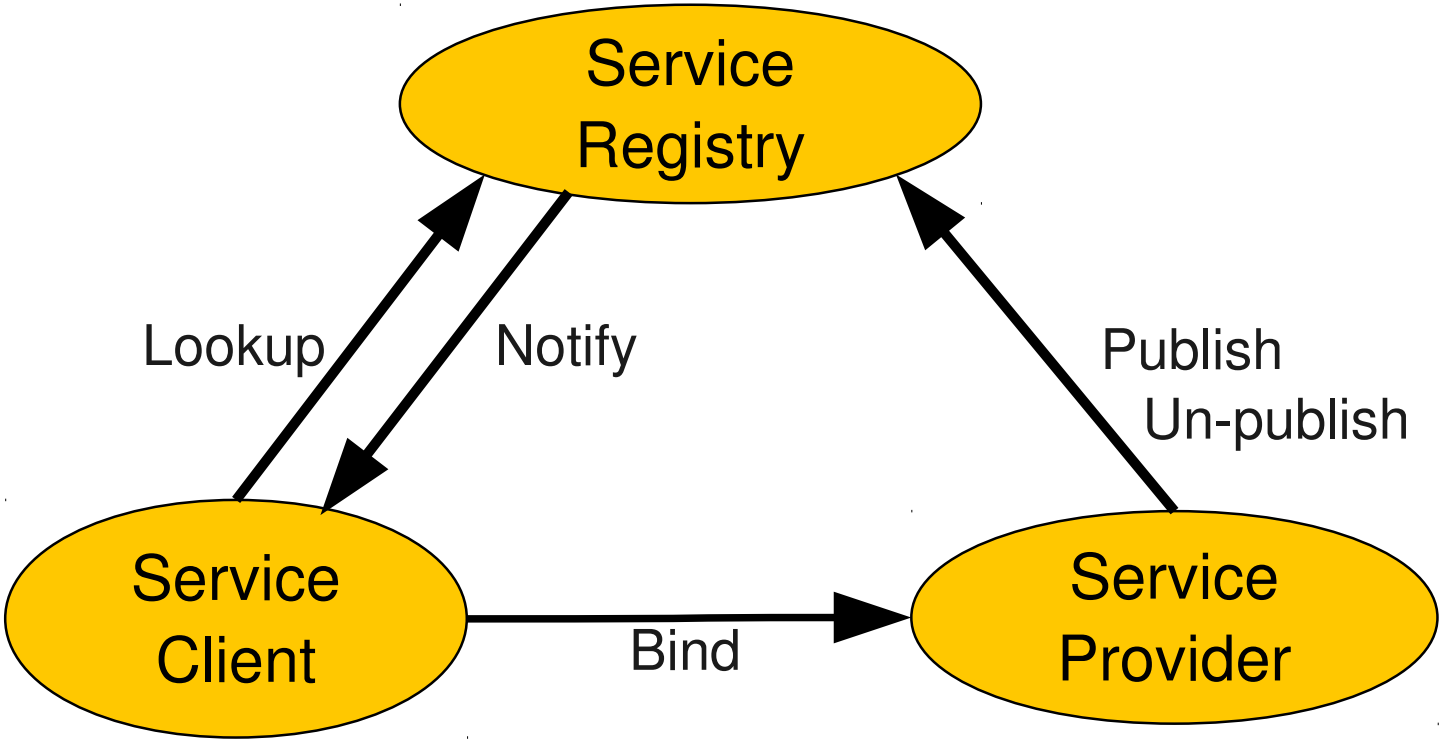
- We propose a framework that eases the development of architecture managers
  - Provides basic services
  - Provides abstractions of the architecture
  - Helps make pertinent decisions on changes
  - Calculates the cost of reconfigurations
  - Can be used to create specialized managers
    - E.g., Minimize footprint, adapt to new requirements, high availability, user context, healing…

How things work
(in our world)

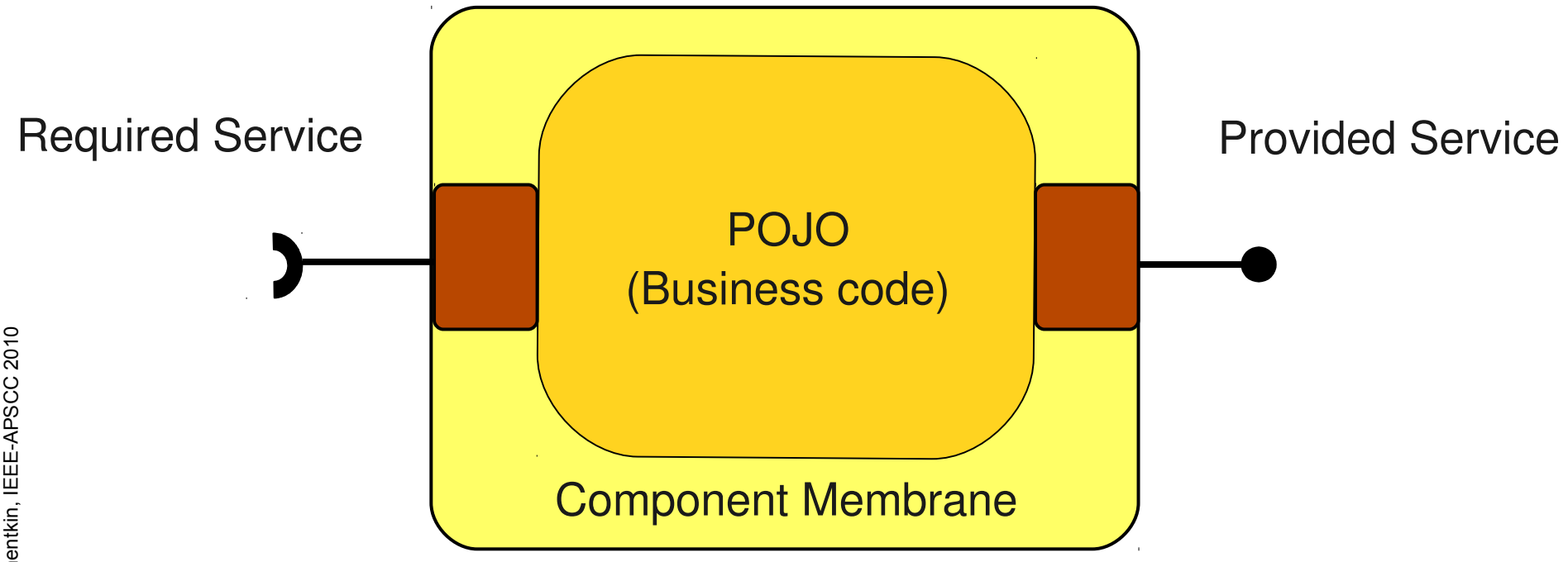
Service Oriented Computing



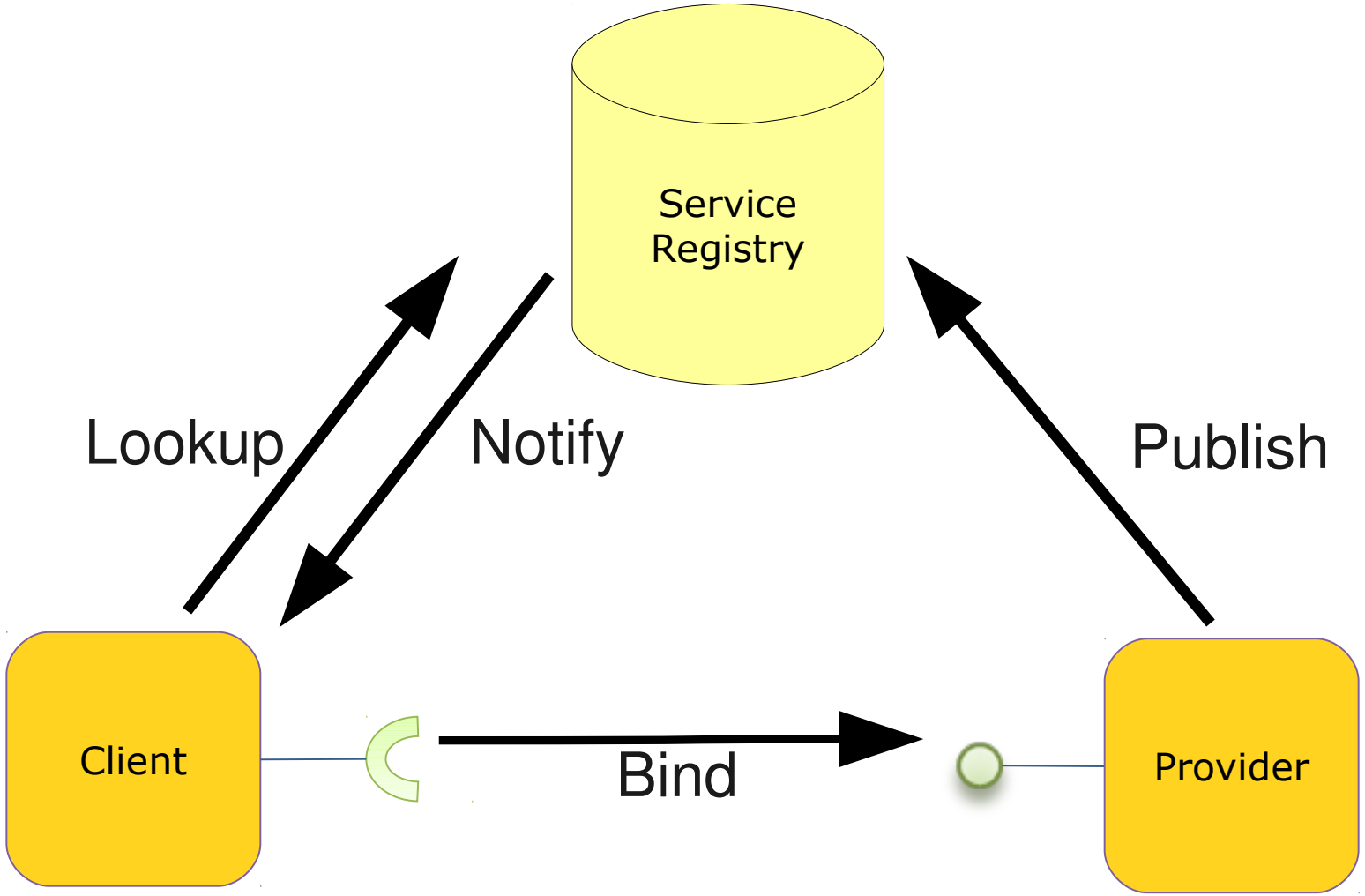
Dynamic Service Oriented Computing



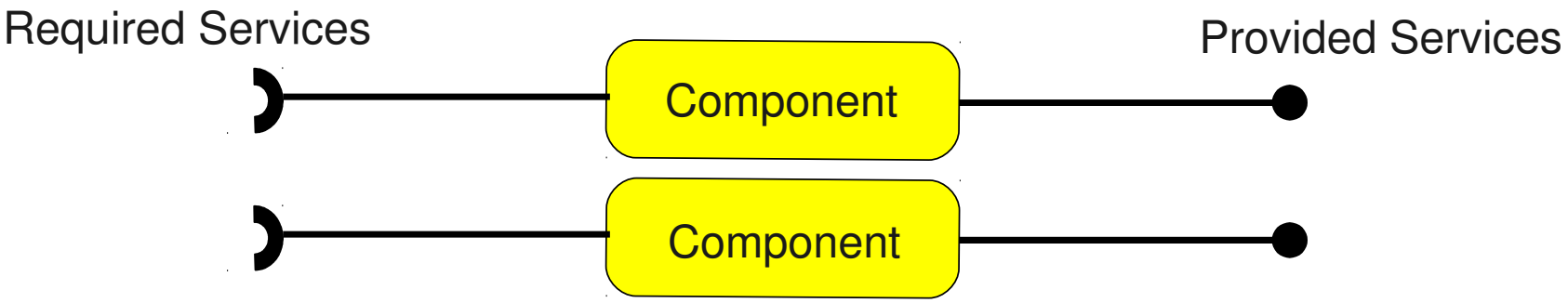
Service-Oriented Components



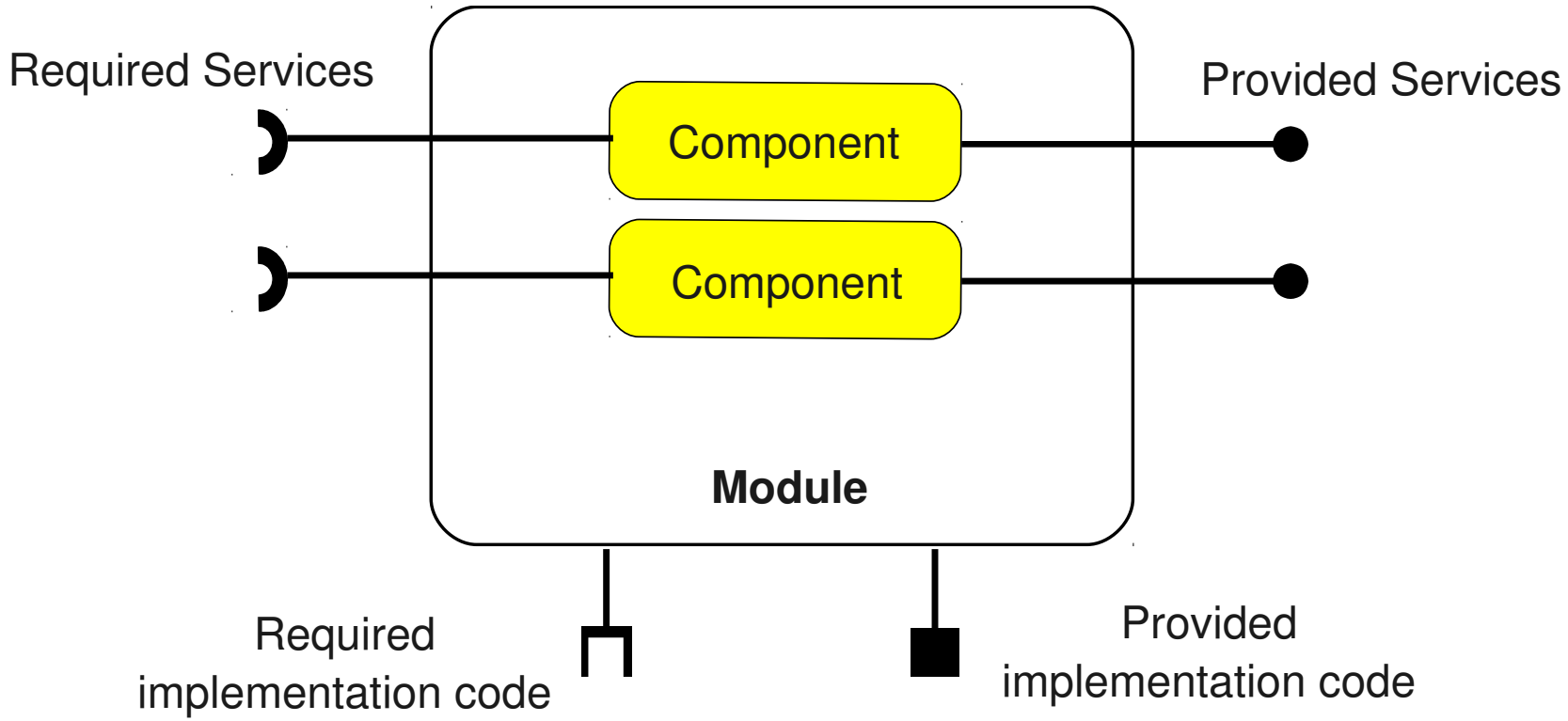
Service-Oriented Components



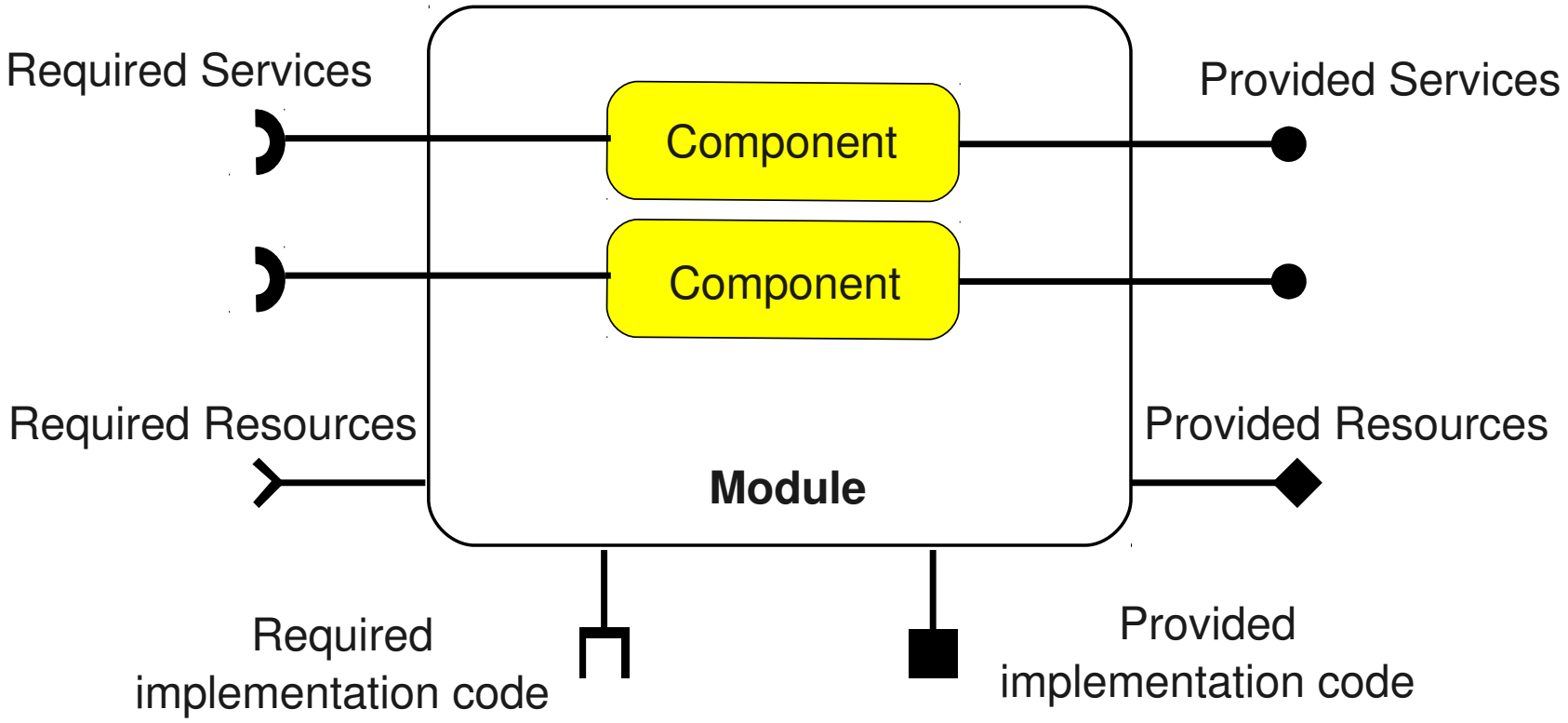
Dependencies: modules and components



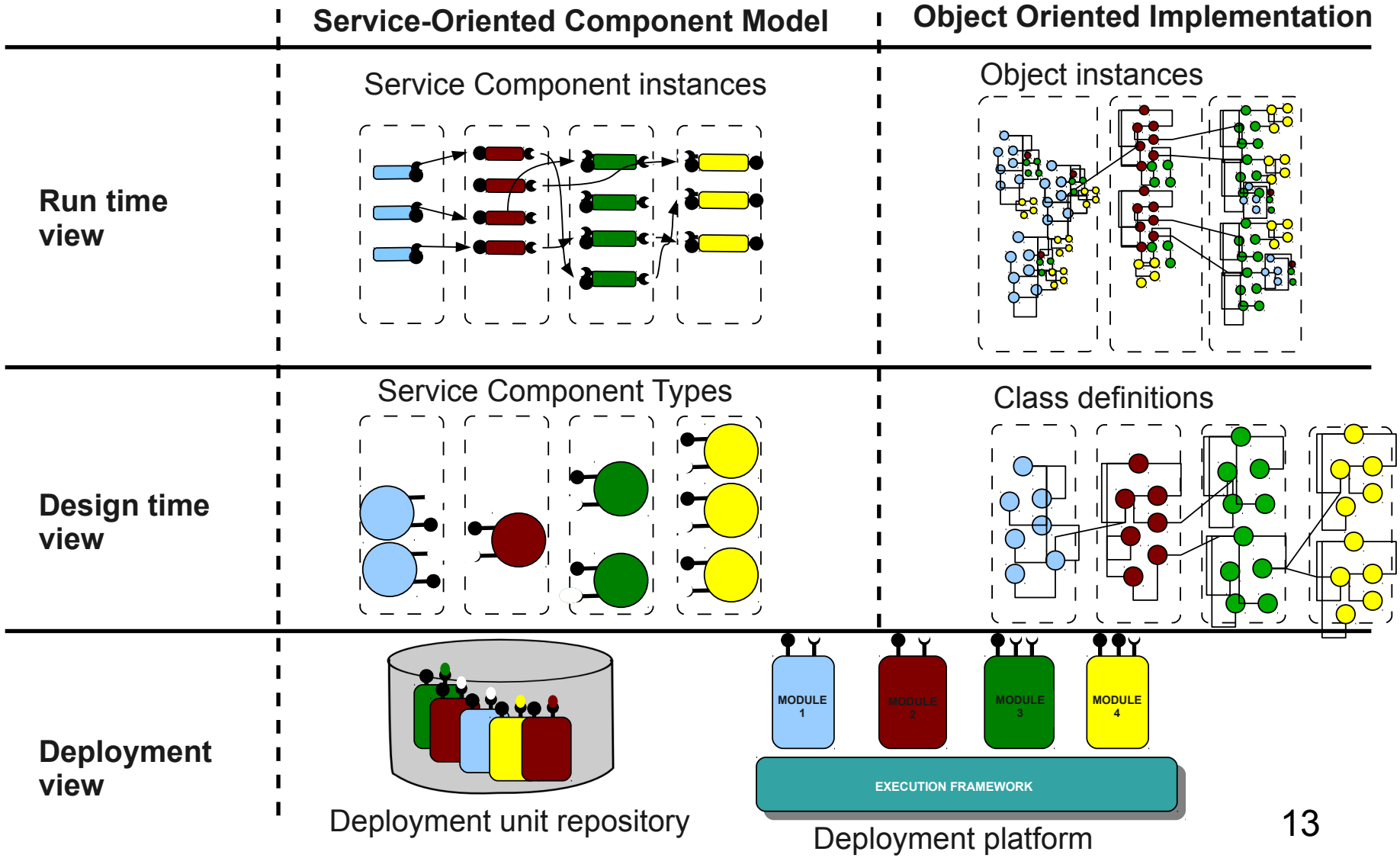
Dependencies: modules and components



Dependencies: modules and components



Service-Oriented Component abstraction levels



What do we do with all of this?

Our approach

- **model@run.time**
  - Architecture model based on dependencies (**Graph**)
- **Management framework**
  - Exports the application model
  - Calculates the cost of a reconfiguration
    - Based on dependency information
  - Proposes management services
    - Repository access, Remote services, Resource management, Application monitoring, …

Model @ runtime: Why analyze dependencies?

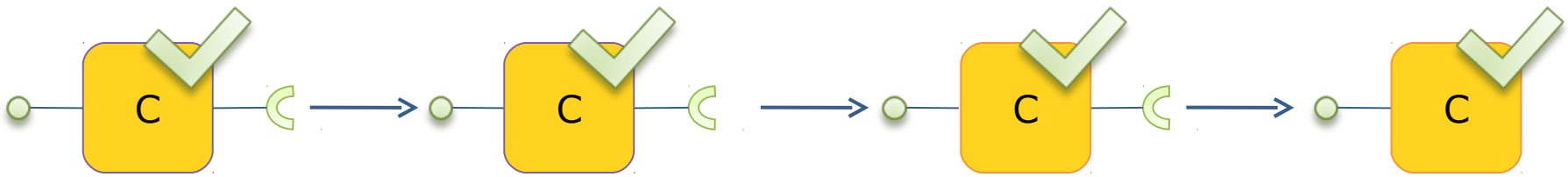
- Primary constraint for reconfigurations
  - Required for installing, instantiating, executing software
- Affect component lifecycle
- Complicate uninstallation
  - E.g.: We want to reduce footprint and not break the application
- Missing dependencies can break the application
  - Halt components, cause state-loss, unavailability, …
- For Centralized Applications
  - (i.e., single memory space)

Impact of a dynamic reconfiguration

- We use the dependency graph to calculate impact
  - Modules stopped (state-loss)
  - Components stopped (possible state-loss)
  - Modules installed and/or restarted
  - Components installed and/or restarted
  - Bindings and re-bindings
- Rollback and recovery not considered

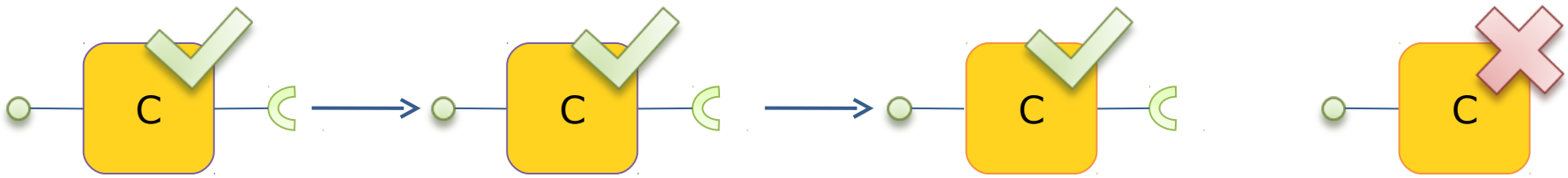
Example: Domino effect

- All components running



Example: Domino effect

- One component stops



Example: Domino effect

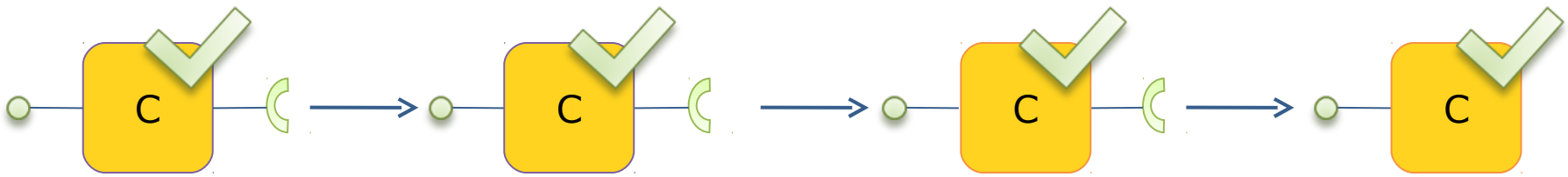
- All components are affected and stopped

Example: Domino effect

- The component becomes available again

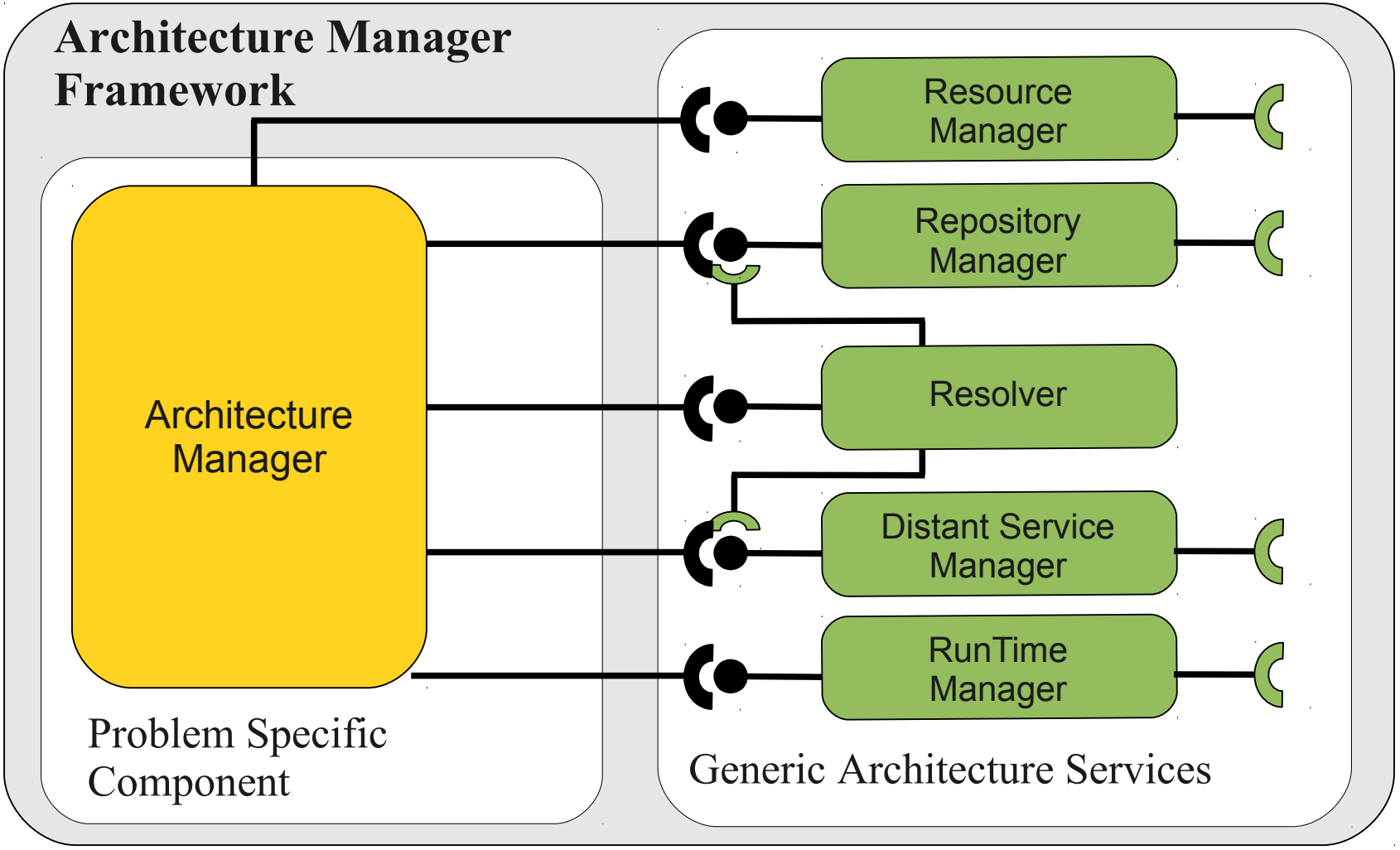
Example: Domino effect

- All components run again

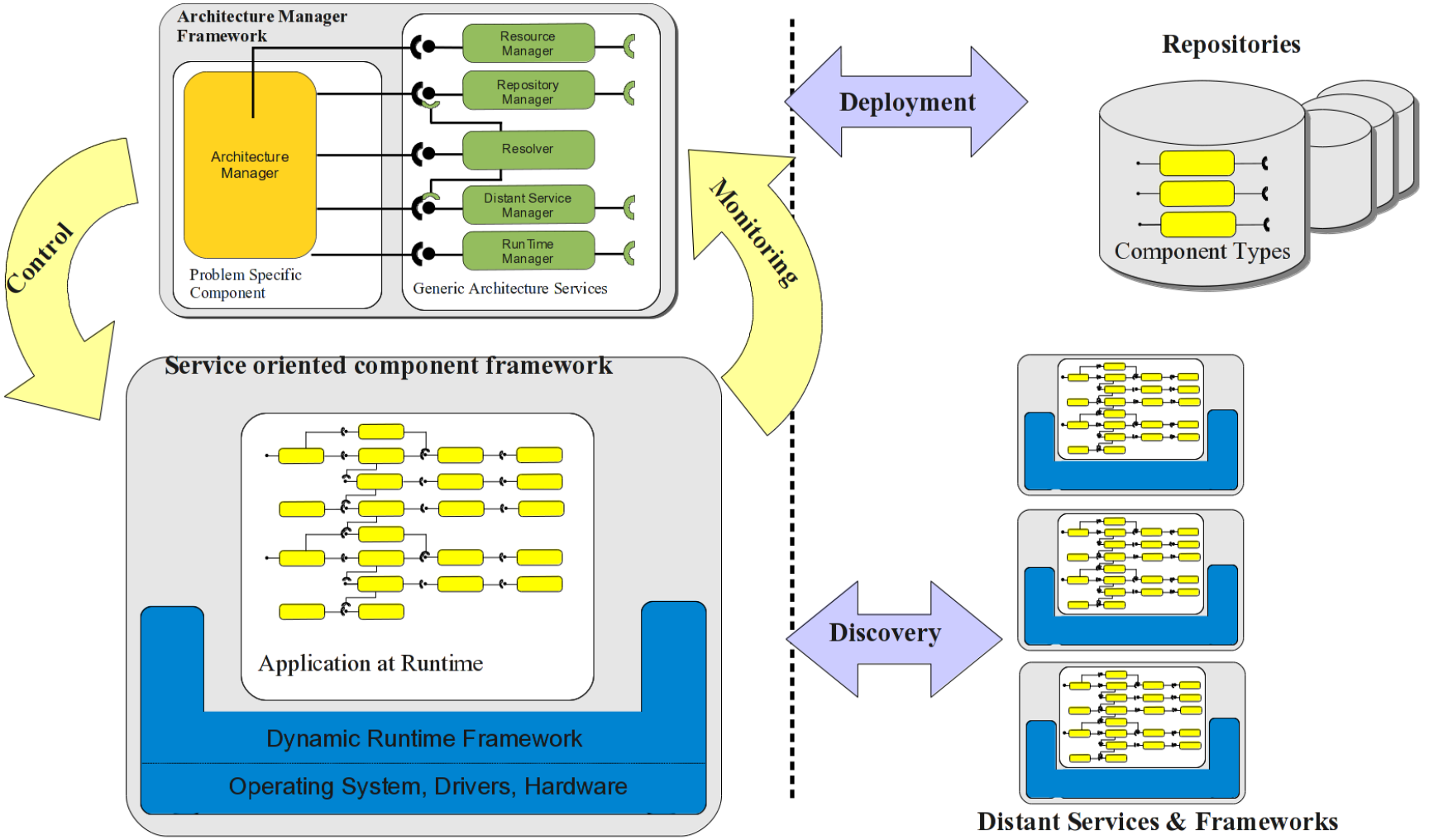


Our prototype

Framework overview



Framework overview: big picture



Implementation details

- Based on the OSGi Service Platform
- Makes extensive use of other projects
  - Apache Felix
  - Apache iPOJO
  - OW2 Chameleon - ROSE
  - OW2 JonAS
  - Eclipse P2
  - SIGAR (SpringSource)
  - ...

Final remarks

Conclusions

- Framework for handling and understanding dynamism in service oriented component platforms
  - Run time impact of dynamic reconfigurations
- We provide the basic mechanisms for manipulating an application's architecture.
- More "intelligent" features can be implemented on top
- The project will be open-sourced on the OW2 JOnAS project in the (near) future.

Perspectives

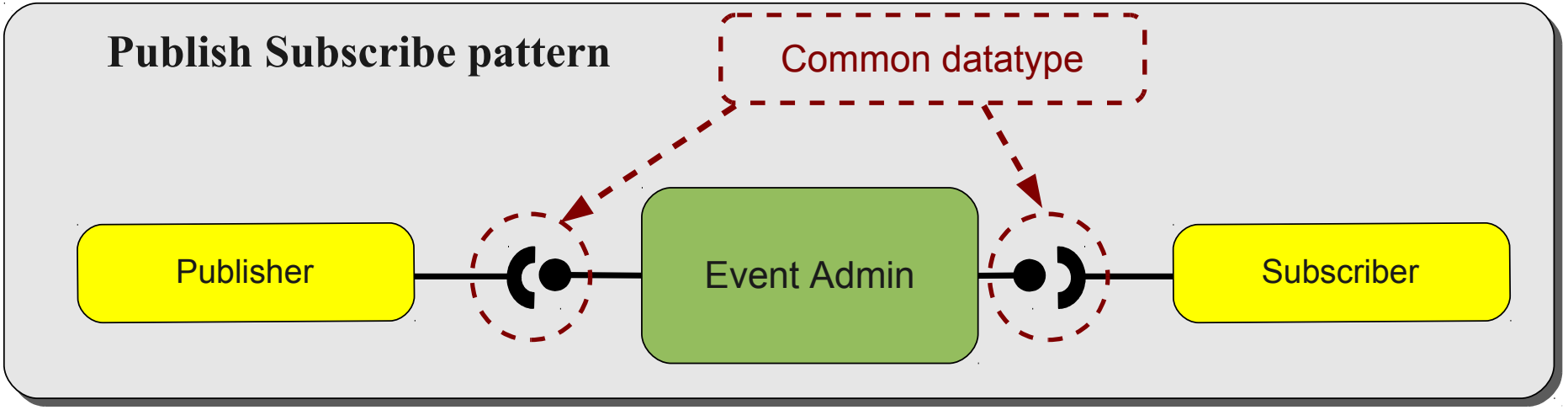
- Define *reactive* properties (application reflexes)
  - Because some actions are not controlled by the application or the manager (eg., devices, remote services)
- Use a *Reference* or *Abstract* architecture
  - To enforce and/or validate the architecture
  - To provide autonomic architecture adaptation and evolution (e.g., based on QoS)

Questions?

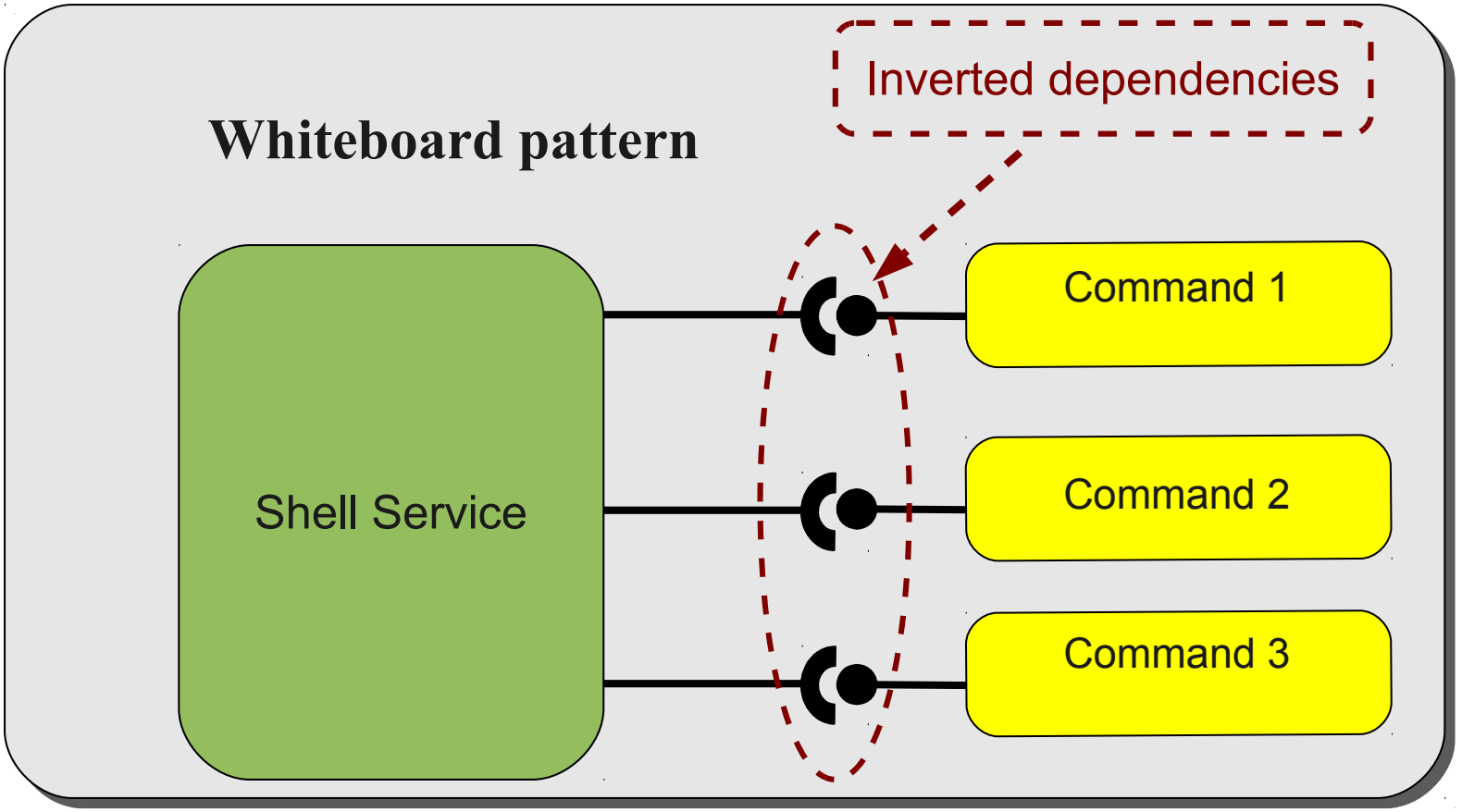
OSGi Dependency classification (related to impact)

Static	Package, Stale references, Dynamic-import, Fragments, Extension points, Handler
Dynamic	Services, Publish-subscribe pattern, Whiteboard pattern
Either	Resources, Extender pattern, Configuration

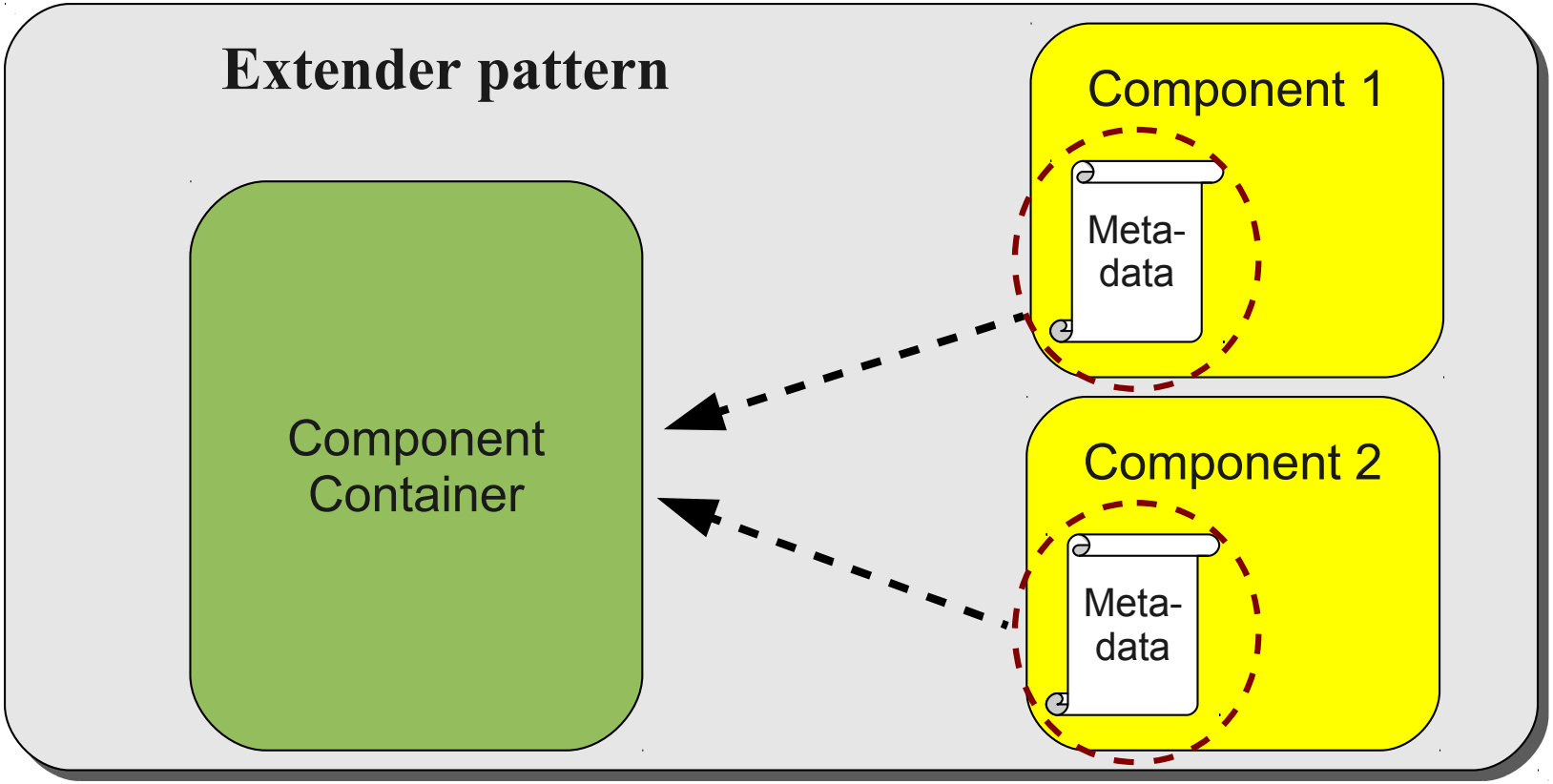
Dependencies: design-pattern



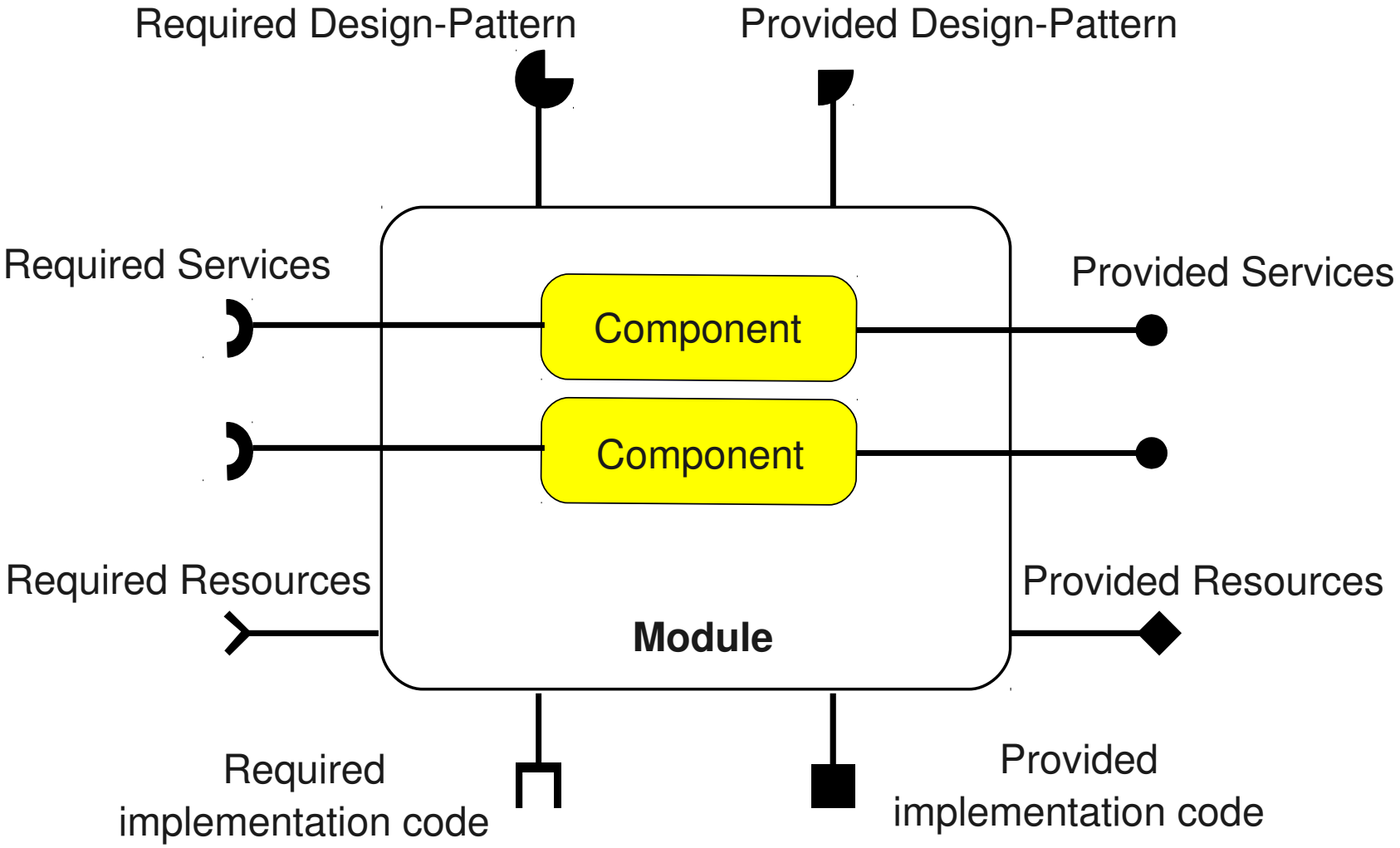
Dependencies: design-pattern



Dependencies: design-pattern



Dependencies: modules and components



Conséquences (4)

- Arrêts en cascade dans le cas d'une composition
  - Effet domino