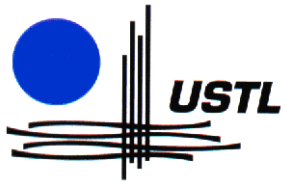
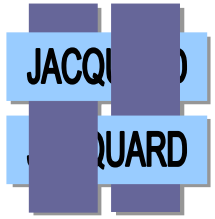




Integrating concerns into a software architecture

Anne-Françoise Le Meur,

Olivier Barais, Laurence Duchien,

Jacquard INRIA project, LIFL, University of Lille, France

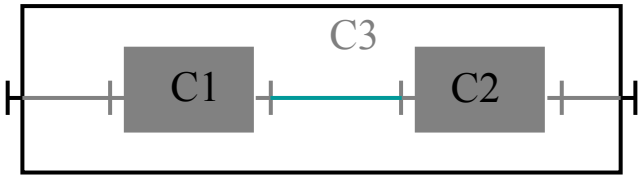
Julia Lawall

DIKU, University of Copenhagen, Denmark

Context : Software Architectures

- Main concepts :

  "...an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections. "

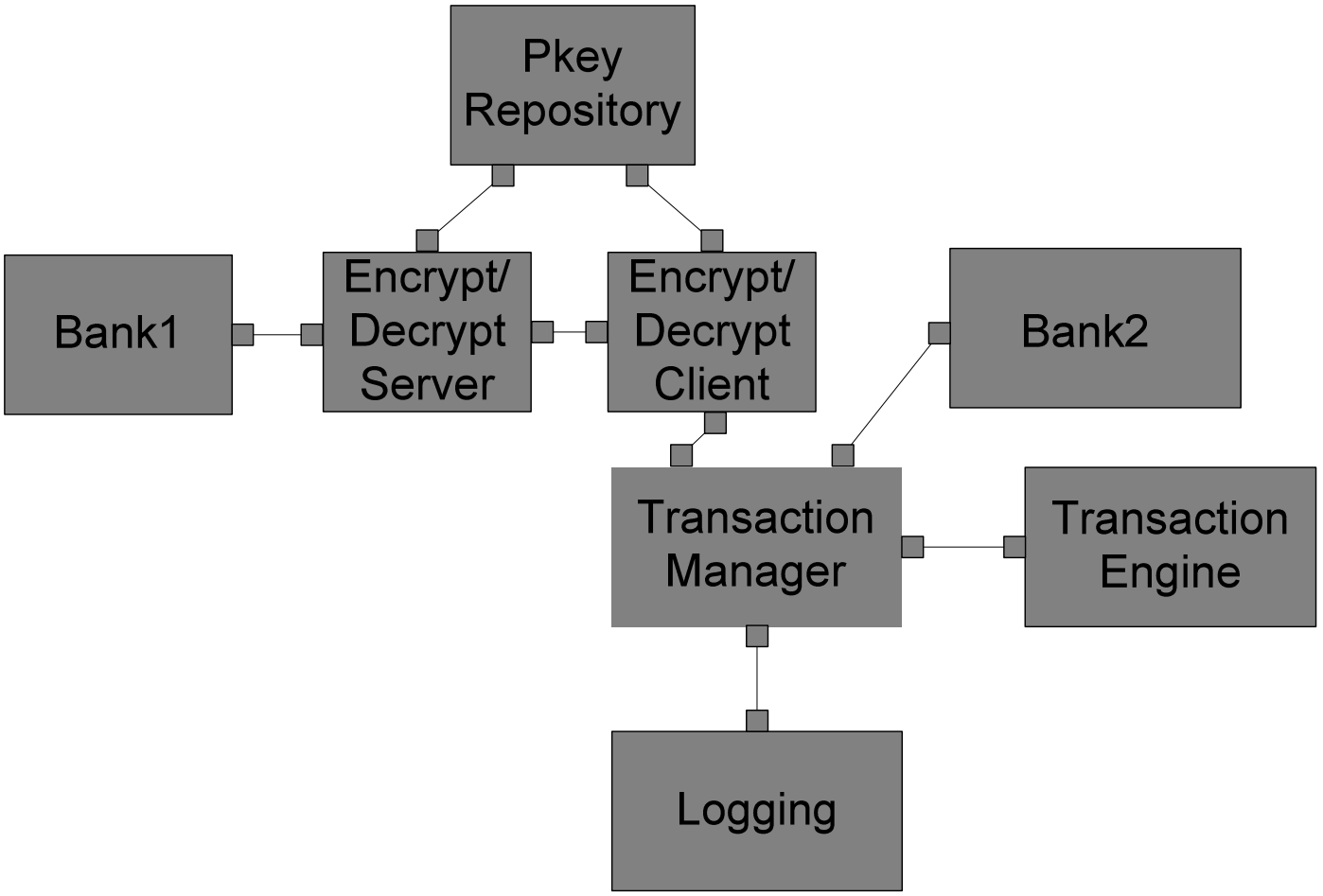


Hayes-Roth, 1994

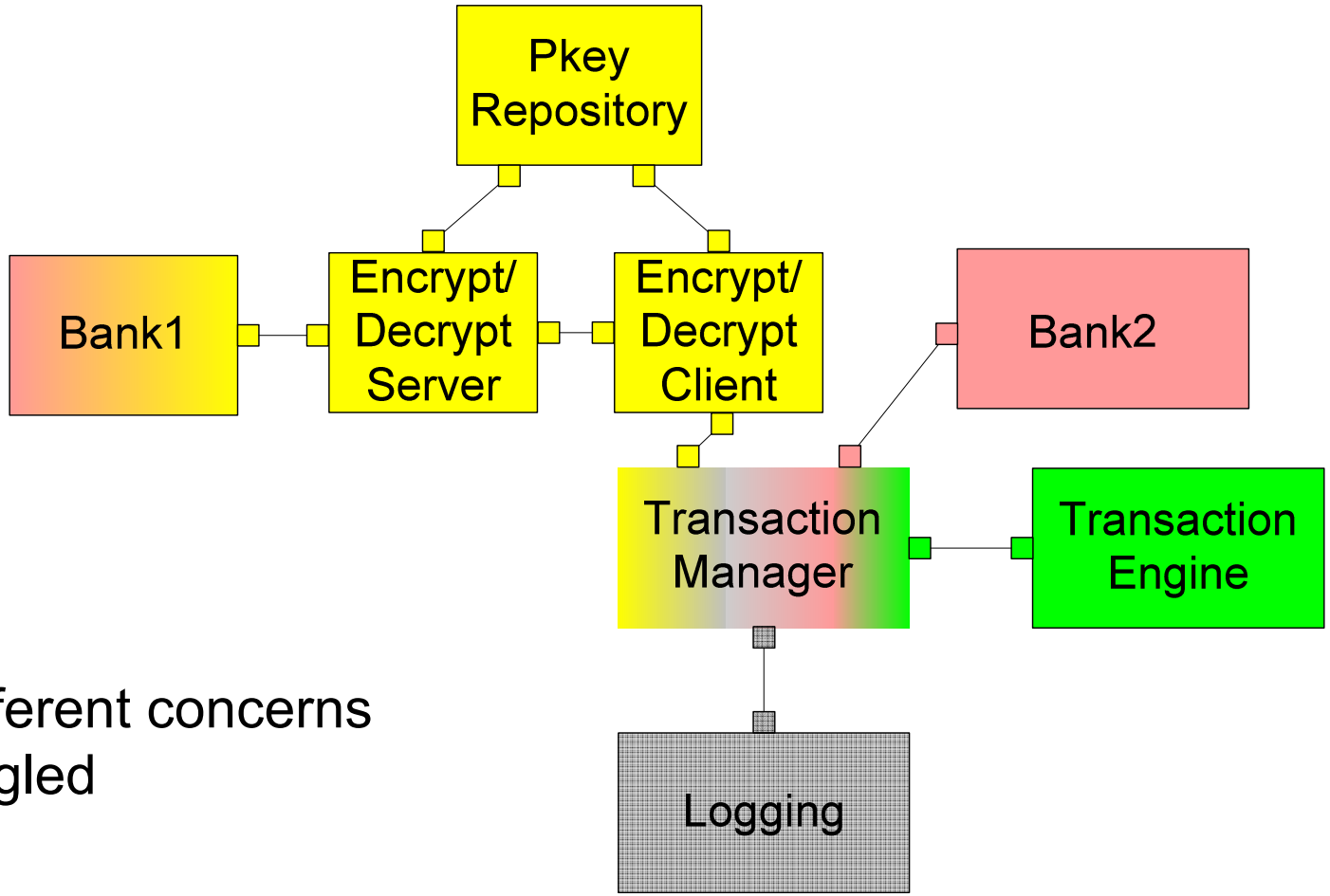
for the ARPA DSSA program

- Benefits
  - understanding (high level abstraction)
  - construction (reuse, code generation)
  - verification
  - evolution (invariants)

Separation of concerns in software architectures

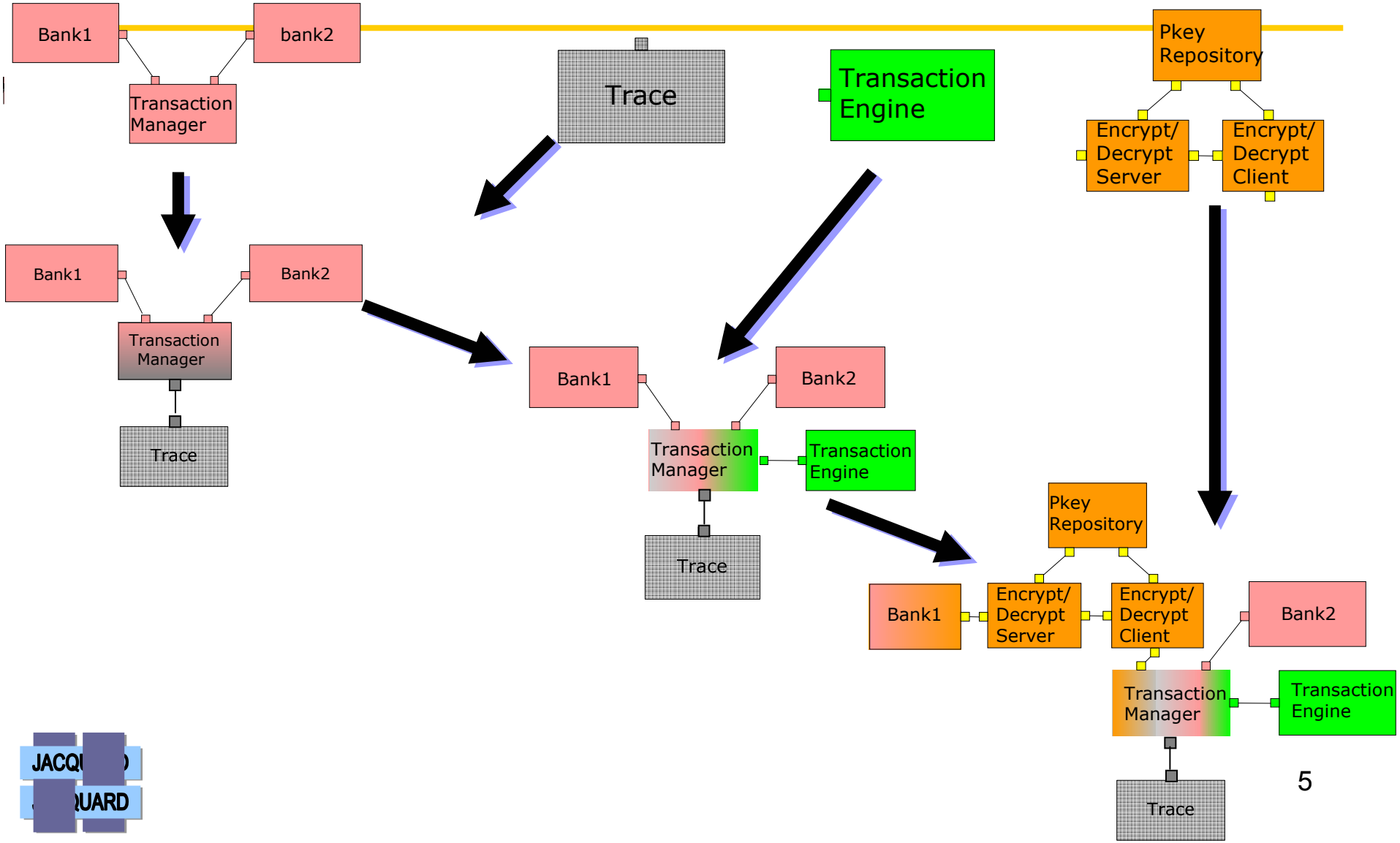


Separation of concerns in software architectures



The different concerns are tangled

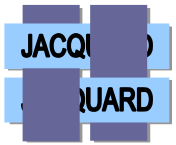
Integrating concerns incrementely



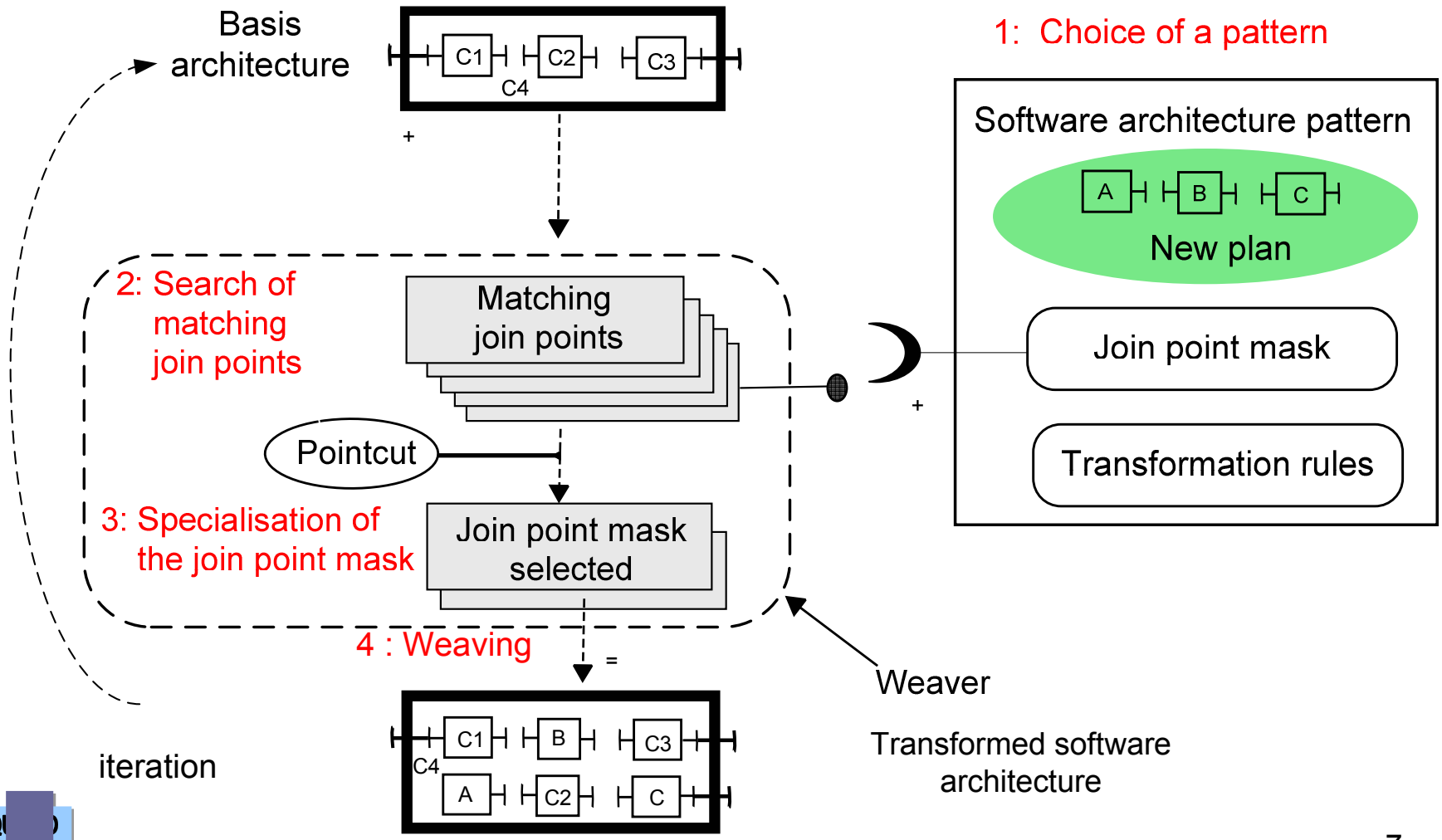
Problems

- Difficult to build a software architecture in several steps
  - Manual and tedious operation (*crosscut*)
  - Error prone
  - No verification between steps
  - No unit to modularize a crosscutting concern and specify its integration

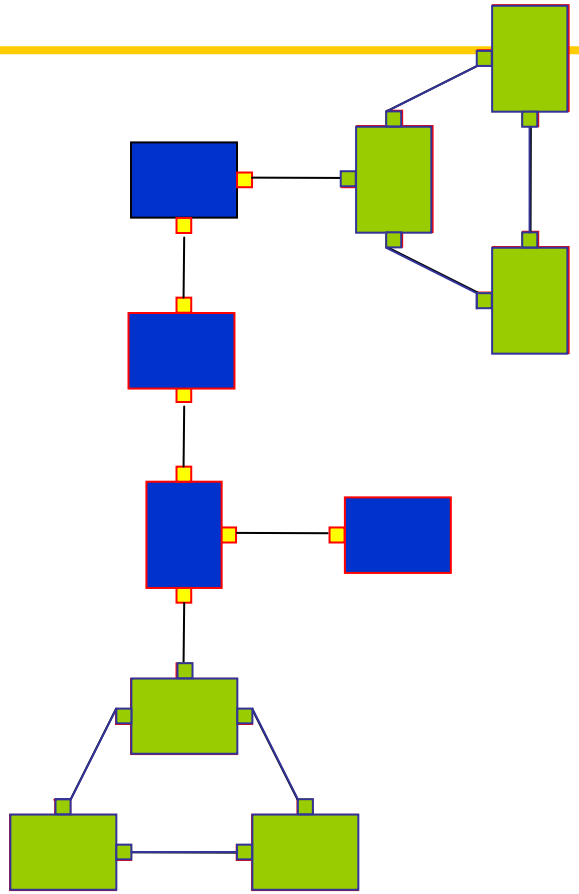
➡ TranSAT : A framework to specify the incremental integration of concerns into a software architecture
- Definition of a new reusable entity modularizing the concept of concern : *a software pattern*
- Integration by transformations



Overview of TranSAT

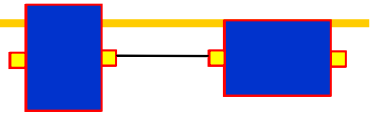


Software architecture pattern

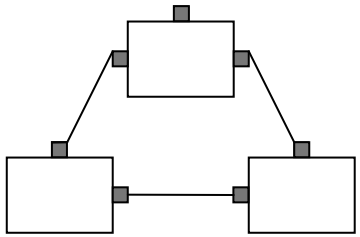


Basis software architecture

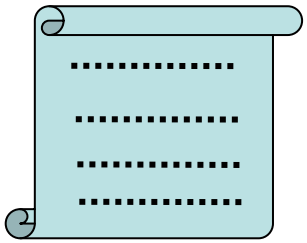
New plan

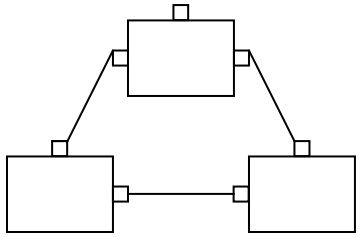


Join point mask



Transformation rules





Software architecture pattern

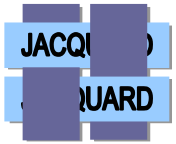
A language to specify transformations

- Defined from the join point mask and the new plan.
- Two kinds of rules:
  - The introduction primitives (intra-component modifications)

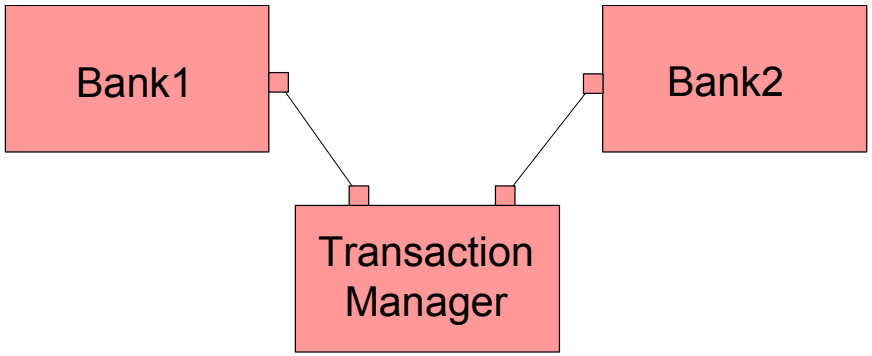
    Modify the specification of a component

    - Structural modifications of a component => its behavioral modification
      - Structural : operation and port addition
      - Behavioral : flow of execution modification
    - Introduction = (structural + behavioral) modifications
  - The reconfiguration primitives (inter-component modifications)

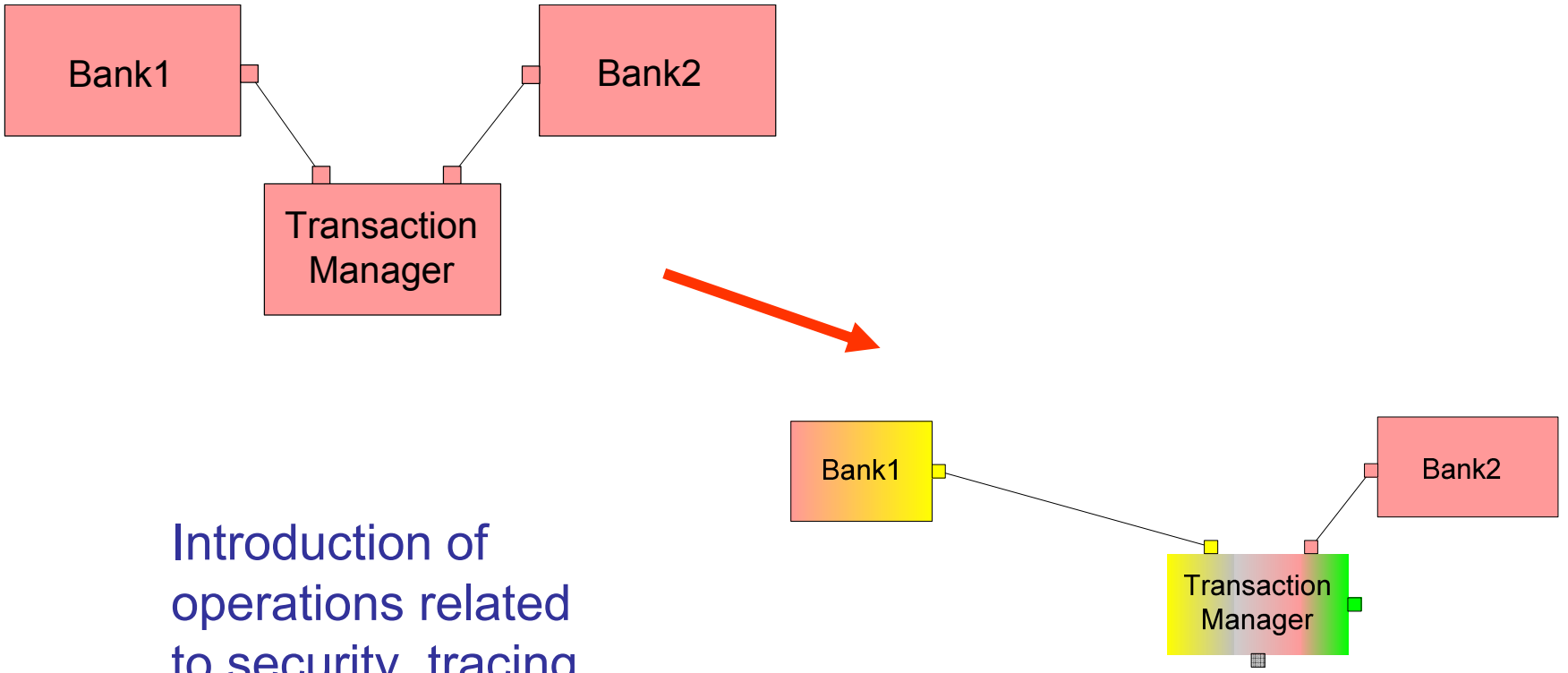
    Modify the software architecture configuration



The transformation rules

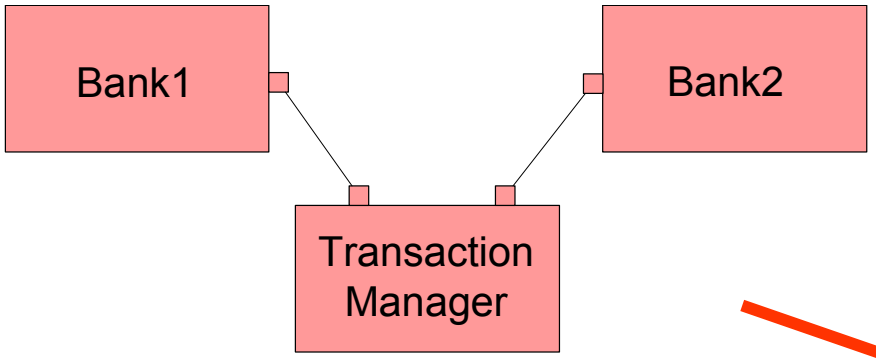


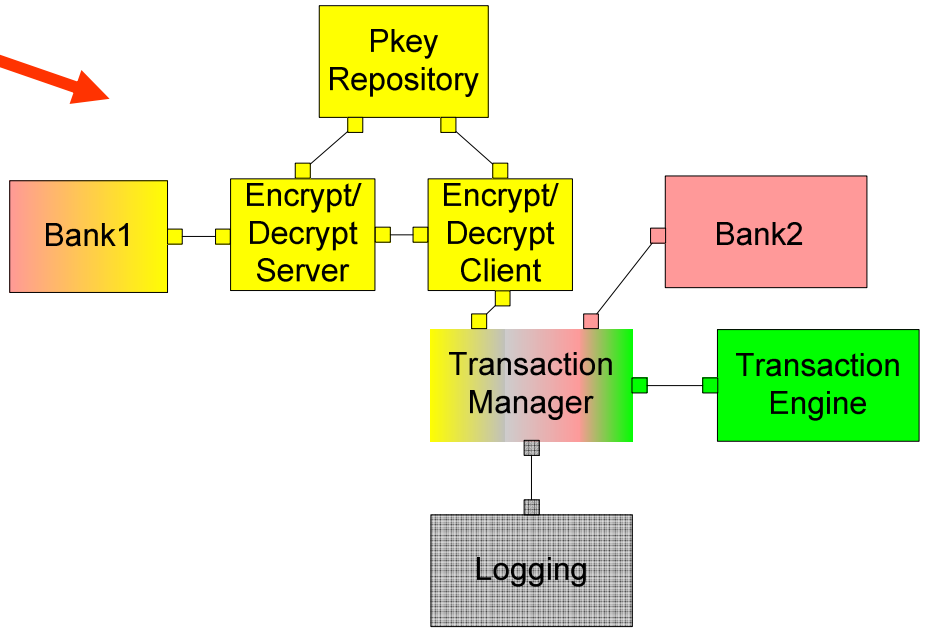
The transformation rules



Introduction of operations related to security, tracing, atomicity concerns

The transformation rules





Introduction +

Reconfiguration

Introduction primitives

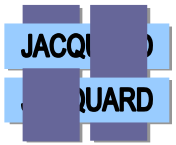
	Port	Operation
Create	`Port` *Pr* **`in`** *Cp*	`Operation` *Or* `= op` **`in`** *Pr* `Operation` Or_1 `= op` **`replace`** Or_2
Destroy	*Pr*`.destroy()`	`N/A`
Move	`N/A`	*Or*`.move(`*Pr*`)`

Cp: ComponentRef; Pr: PortRef; Or: OperationRef;
op ::= *Or* | inverse(*Or*); N/A: Not applicable;

Reconfiguration primitives

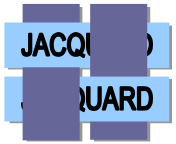
	Component	Binding	Composite
Create	N/A	Binding *Br* ={*Pr*$_1$,*Pr*$_2$}	Composite *Cr* Composite *Cr*$_1$ **in** *Cr*$_2$
Destroy	N/A	*Br*.destroy()	N/A
Move	*Cp*.move(*Cr*)	N/A	*Cr*$_1$.move(*Cr*$_2$)

Cp: ComponentRef; Cr: CompositeRef; Pr: PortRef;
Br: BindingRef; N/A: Not applicable;



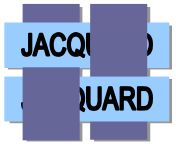
Transformation coherence

- Static analyses of the pattern (once and for all)
  - check of the coherence of the new plan, the join point mask and the transformation rules
  - verification of the impact of the transformation on the architecture coherence with respect to its structure
- Dynamic analyses of the transformation (for each integration)
  - verification of the structural compatibility of the modified or created connexions
  - verification of the behavioral compatibility of the new or modified components assemblies
  - verification of the impact of the transformation on the architecture coherence with respect to its behavior



Transformation coherence

- Static analyses of the pattern (once and for all)
  - check of the coherence of the new plan, the join point mask and the transformation rules
  - verification of the impact of the transformation on the architecture coherence with respect to its structure
- Dynamic analyses of the transformation (for each integration)
  - verification of the structural compatibility of the modified or created connexions
  - verification of the behavioral compatibility of the new or modified components assemblies
  - verification of the impact of the transformation on the architecture coherence with respect to its behavior



Strutural impact

- Definition of the operational semantics of each primitive

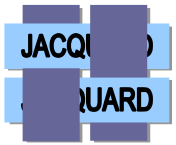
Ex : `Or.move(Pr)`

$$\frac{Or \in J \qquad \pi_2 = \pi_3}{\begin{array}{l}\rho[Or \mapsto (\pi_1, \zeta_1), Pr \mapsto (\pi_2, \zeta_2), \pi_1 \mapsto (\pi_3, \zeta_3)], J, P \vdash Or.\texttt{move}\,(Pr) \\ \quad \rightarrow \rho[Or \mapsto (Pr, \zeta_1), Pr \mapsto (\pi_2, \zeta_2 \cup \{Or\}), \pi_1 \mapsto (\pi_3, \zeta_3 - \{Or\})], J - \{Or\}, P\end{array}}$$

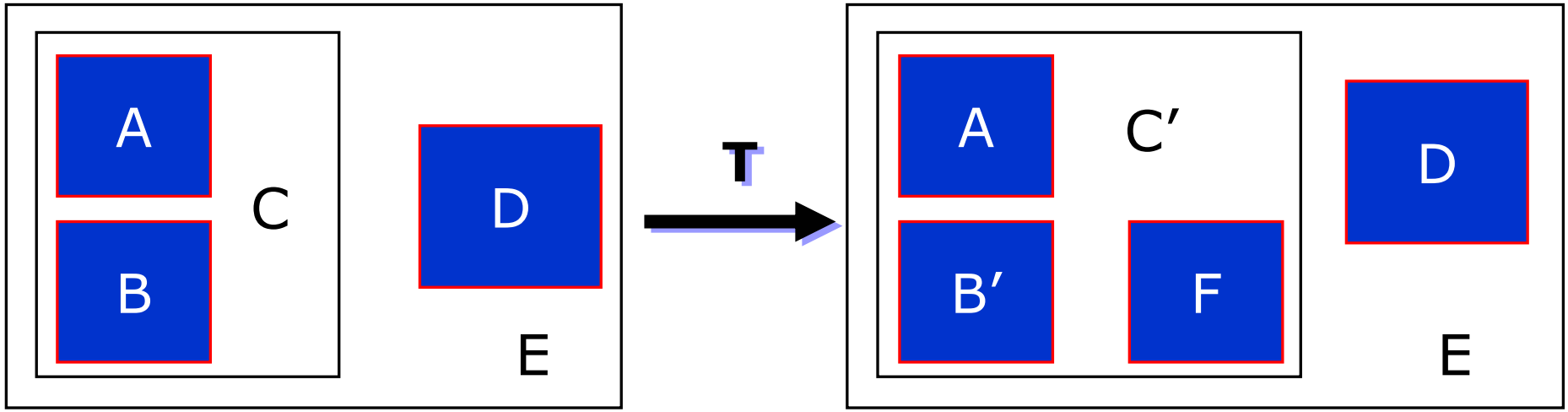
- Containment analysis
  - Each element must have some subelements
  - Usefulness of each transformation rule
  - Simulation of the transformations

Transformation coherence

- Static analyses of the pattern (once and for all)
  - check of the coherence of the new plan, the join point mask and the transformation rules
  - verification of the impact of the transformation on the architecture coherence with respect to its structure
- Dynamic analyses of the transformation (for each integration)
  - verification of the structural compatibility of the modified or created connexions
  - verification of the behavioral compatibility of the new or modified components assemblies
  - verification of the impact of the transformation on the architecture coherence with respect to its behavior



Impact on the behavior



- Are the behavioral contracts of A, B', F compatible?
- Is the behavior of C' bisimilar to the one of C?

Summary

- TranSAT : a model transformation approach
  - The pattern : a new entity to modularize a concern in a software architecture
  - A transformation process specified and verified
    - A model to specify constraint on the integration context of the pattern
    - A specific language to specify the transformations
    - "Write once integrate everywhere"
  - Tools (static analyses + search -> DROOLS, transformation -> AGG)
- Perspectives
  - Validation on industrial examples
  - Generalization of the approach to target other ADLs
  - Semantic join point mask
  - …