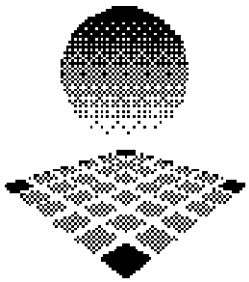


UNIVERSITÉ MONTPELLIER II
SCIENCES ET TECHNIQUES DU LANGUEDOC

Automating Fine Concurrency Control in Object-Oriented Databases

Carmelo MALTA
José MARTINEZ

OUTLINE

HYPOTHESIS ON THE DATABASE

FOUR PROBLEMS

A SOLUTION

- AT COMPILE-TIME
- AT LINK-TIME
- AT RUN-TIME

A COMPARISON

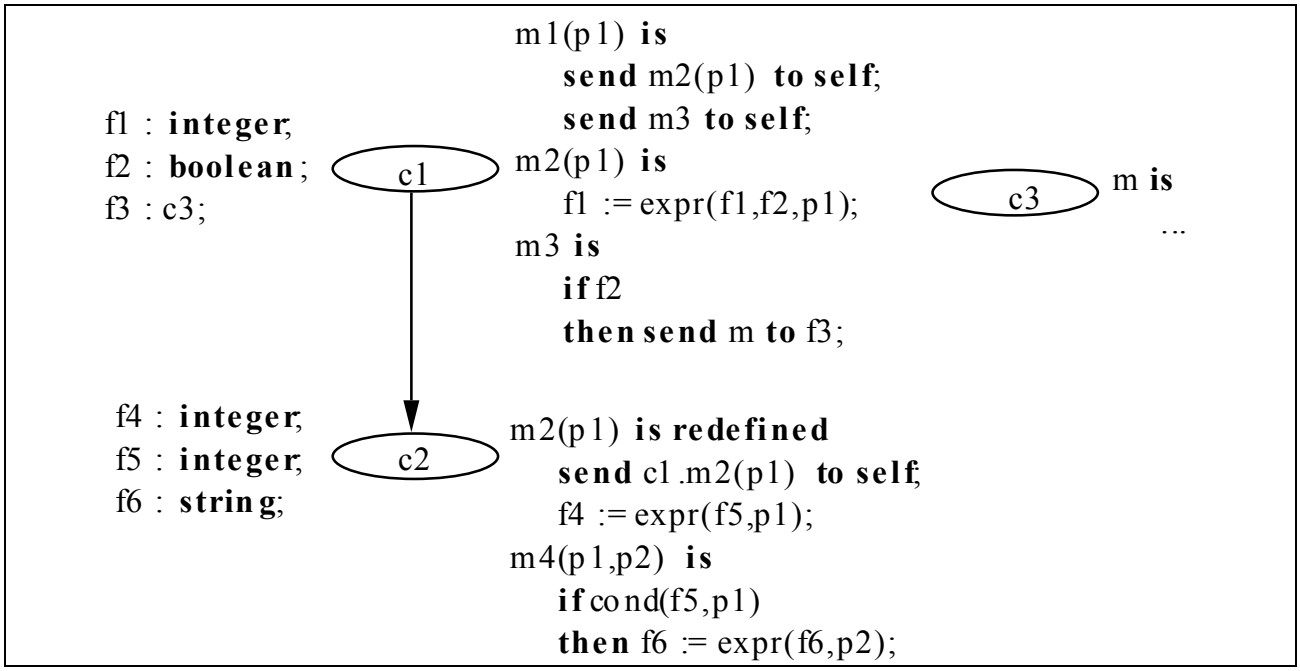
CONCLUSION

HYPOTHESIS ON THE DATABASE

+ on the data:
  - **classes** (but no meta-classes)
  - (multiple) **inheritance** (inclusion and constraint)
  - **mono-instanciation**

+ on the methods:
  - **encapsulation** (mandatory)
  - **overriding**
  - **late binding**

FOUR PROBLEMS



+ when determining commutativity of methods
  - (m_1, m_2, m_3) in c_1
  - (m_1, m_2, m_3, m_4) in c_2
  - (m_1, m_2, m_3, m_4, ...) in c_n

+ repeated controls
  - controlling m_1, next m_2, then m_3 when using m_1 in c_1

+ lock escalation
  - m_1 just needs READ access
  - but then m_2 requires WRITE access

+ *pseudo*-conflicts
  - m_2 and m_4 in c_2 should be allowed concurrently

A SOLUTION

At compile-time

analysis of the code of the methods

At link-time

construction of the late binding resolution graph (for self-directed messages only)

using this graph for calculating *transitive access vectors*

translating access vectors into mere access modes

At run-time

using these access modes in the locking protocol “as usual”

AT COMPILE-TIME

+ determining DAV (Direct Access Vectors)

For each method, determine which instance variables are respectively read and/or written.

+ determining DSC (Direct Self-Calls)

For each method, extract the names of the direct messages which are sent to **self** (i. e., **send** M **to self**).

+ determining PSC (Prefixed Self-Calls)

For each method, extract the names of the classes and the names of the prefixed messages which are sent to **self** (i. e., **send** C.M **to self**).

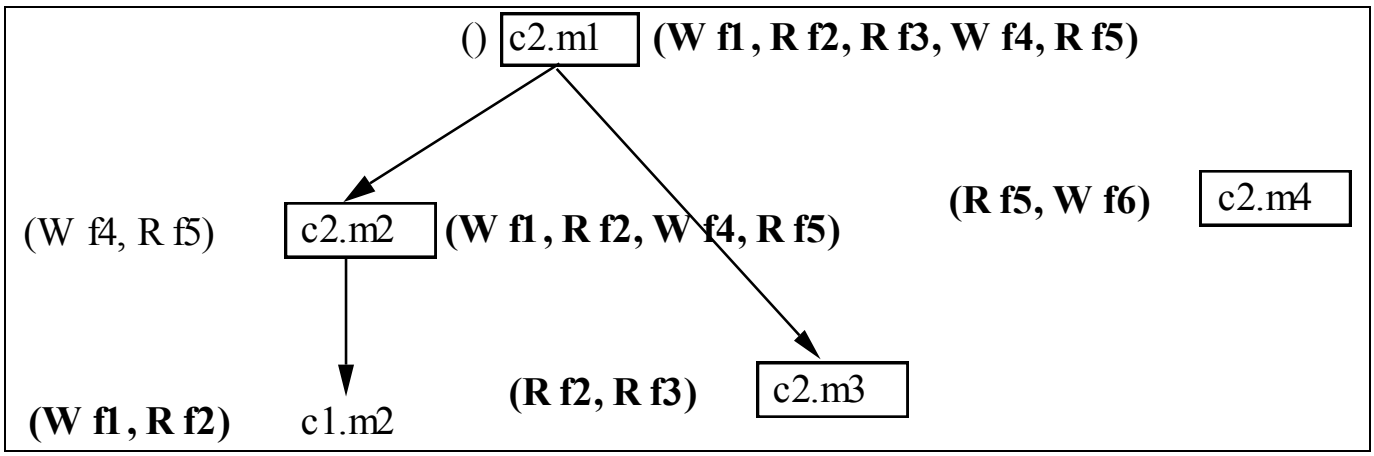
class	method	analysis
c1	m1(p1) **is** **send** m2(p1) **to self**; **send** m3 **to self**;	DAV = () DSC = {m2,m3} PSC = Ø
	m2(p1) **is** f1 := expr(f1,f2,p1);	DAV = (Write f1, Read f2) DSC = Ø PSC = Ø
	m3 **is** **if** f2 **then send** m **to** f3;	DAV = (Read f2, Read f3) DSC = Ø PSC = Ø
c2	m1 **is inherited**	
	m2(p1) **is redefined** **send** c1.m2(p1) **to self**; f4 := expr(f5,p1);	DAV = (Write f4, Read f5) DSC = Ø PSC = {c1.m2}
	m3 **is inherited**	
	m4(p1,p2) **is** **if** cond(f5,p1) **then** f6 := expr(f6,p2);	DAV = (Read f5, Write f6) DSC = Ø PSC = Ø

AT LINK-TIME

+ constructing the late binding resolution graph

$DSC_{c2,m1} = \{m2, m3\}$

$PSC_{c2,m2} = \{c1.m2\}$



Late binding resolution graph for proper instances of class c2

+ calculating TAV (Transitive Access Vectors)

$TAV_{c1,m2} = DAV_{c1,m2}$

$TAV_{c2,m3} = DAV_{c2,m3}$

$TAV_{c2,m4} = DAV_{c2,m4}$

$TAV_{c2,m2} = DAV_{c2,m2} + TAV_{c1,m2}$

$TAV_{c2,m1} = DAV_{c2,m1} + TAV_{c2,m2} + TAV_{c2,m3}$

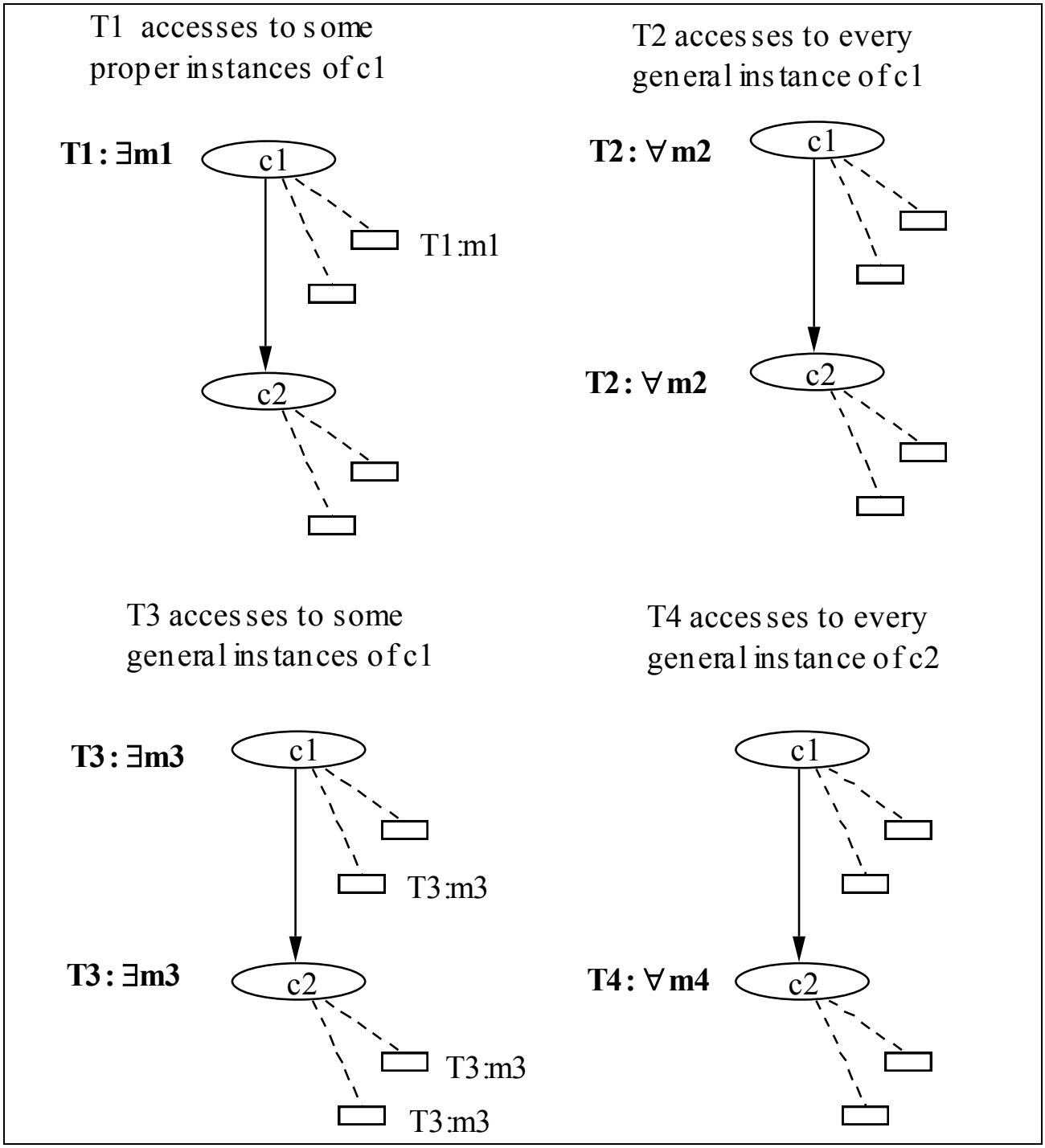
+ Translating vectors into access modes

	m1	m2	m3	m4
m1			**yes**	**yes**
m2			**yes**	**yes**
m3	**yes**	**yes**	**yes**	**yes**
m4	**yes**	**yes**	**yes**	

Commutativity of methods in class c2

AT RUN-TIME

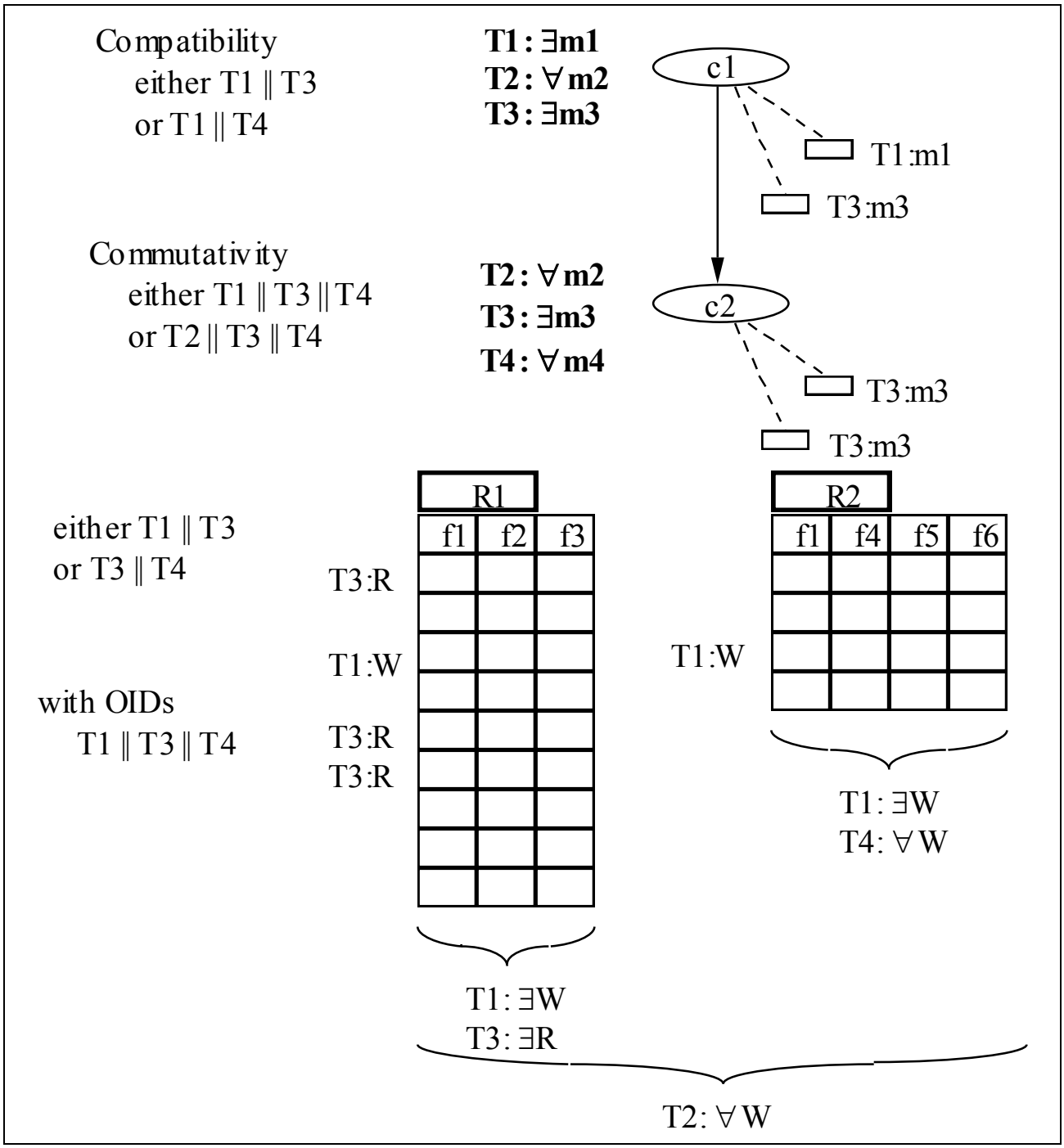
Access modes + Hierarchical locking



A COMPARISON

	m1	m2	m3	m4
M1	yes/no		yes/no	
M2				
M3	yes/no		**yes**	
M4				

Compatibility of methods in class c2



CONCLUSION

+ Commutativity ♠ compatibility □ simple technique

+ This proposition > CC on inheritance graph
  This proposition > CC in relational databases

FURTHER RESEARCHES

+ About composition: multi-level transactions and recovery

+ About the many relationships among objects: inheritance, composition, versioning, composite objects, etc