Distributed Systems

Focus on: Collaborative Platforms, Shared Memory Distributed Algorithms

Outline

Introduction | CSCW | Shared Memory | Distributed Algorithms | Validation | Message Passing | Advances | Conclusion

I. Computer Supported Collaborative Work

II. Shared Memory Theory

III. Distributed Algorithms for Shared Memory Management

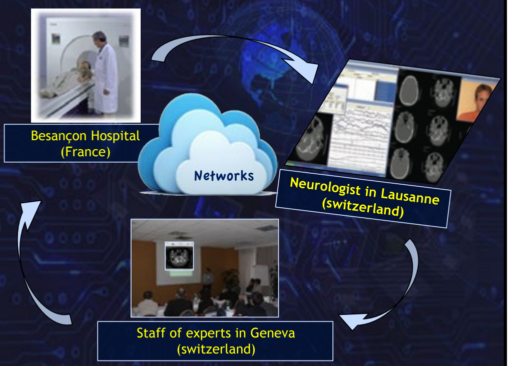
IV. Validations of Distributed Algorithms

V. Implementations on Message Passing Layer

VI. Computer Science Advances in these Domains

Collaboration

Introduction | CSCW | Shared Memory | Distributed Algorithms | Validation | Message Passing | Advances | Conclusion



Collaboration Domain

Introduction | CSCW | Shared Memory | Distributed Algorithms | Validation | Message Passing | Advances | Conclusion

- Internet → a new powerful communication vector
- Improvement of performance
  - Processors level (compression treatments…)
  - Networks levels (flow rates, Quality of Services, …)
- Different domains
  - Remote teaching: teleteaching
  - Tele-Medicine
  - Collaborative editing
  - videoconferencing…

What are the specificities of this kind of applications

Introduction | CSCW | Shared Memory | Distributed Algorithms | Validation | Message Passing | Advances | Conclusion

- Are images the only one medium?
- Is the text used?

Discrete Media

- Does this application use audio?
- And video?

Continuous Media

Two types (groups) of media are distinguished

Outline

Introduction | CSCW | Shared Memory | Distributed Algorithms | Validation | Message Passing | Advances | Conclusion

I. Computer Supported Collaborative Work
- CSCW Domain,
- Distributed Architecture,
- Distributed HCI,
- Example of Applications.

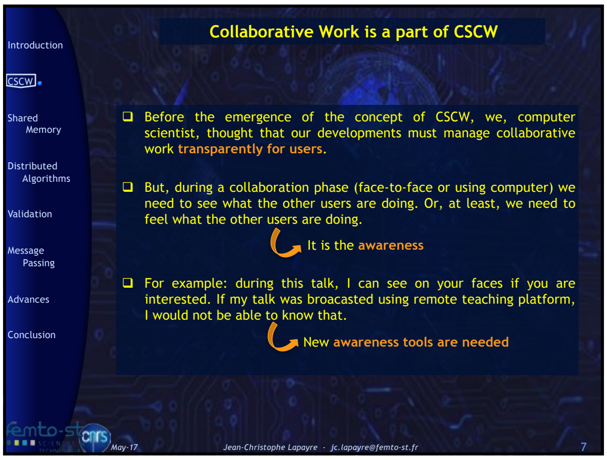
II. Shared Memory Theory

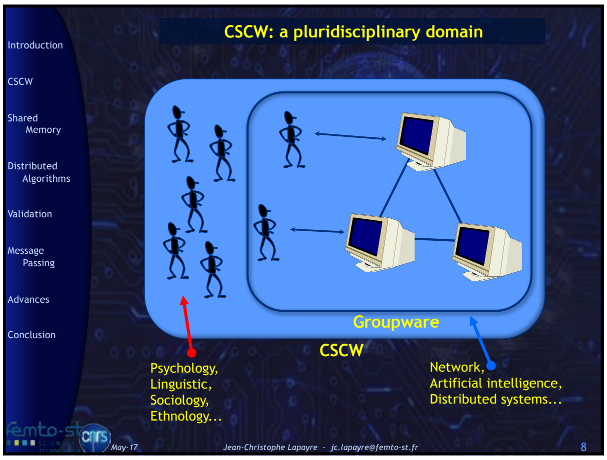
III. Distributed Algorithms for Shared Memory Management

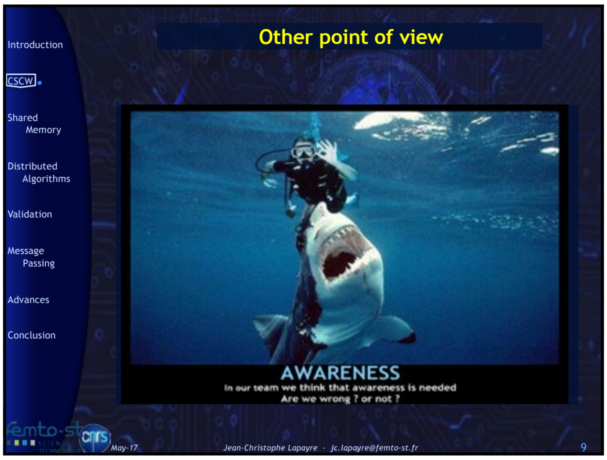
IV. Validations of Distributed Algorithms

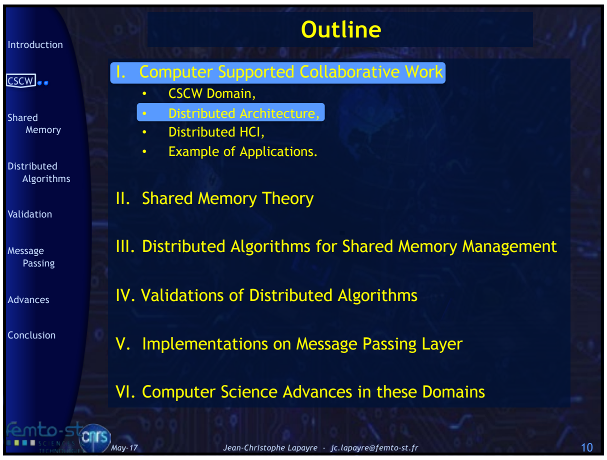
V. Implementations on Message Passing Layer

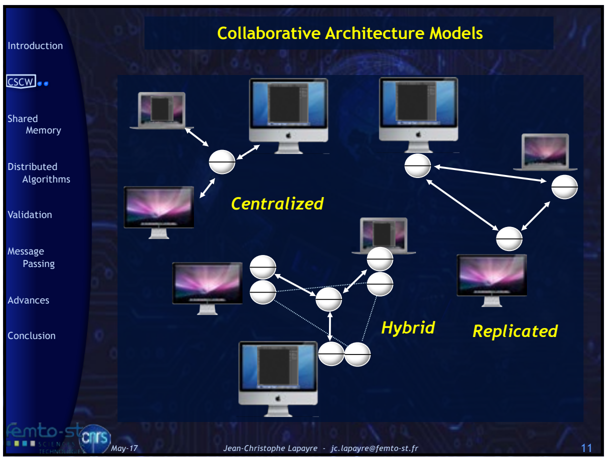
VI. Computer Science Advances in these Domains

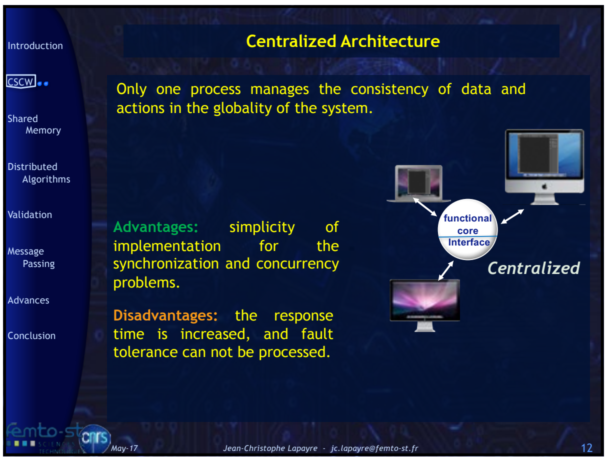

Collaborative Work is a part of CSCW
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Before the emergence of the concept of CSCW, we, computer scientist, thought that our developments must manage collaborative work transparently for users.
But, during a collaboration phase (face-to-face or using computer) we need to see what the other users are doing. Or, at least, we need to feel what the other users are doing.
It is the awareness
For example: during this talk, I can see on your faces if you are interested. If my talk was broacasted using remote teaching platform, I would not be able to know that.
New awareness tools are needed
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
7

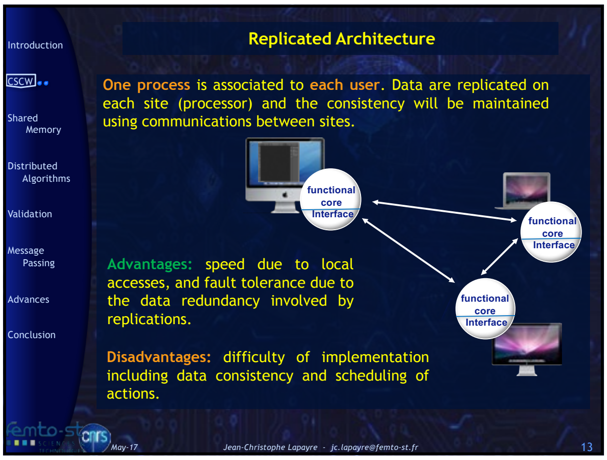

CSCW: a pluridisciplinary domain
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Groupware
CSCW
Psychology,
Linguistic,
Sociology,
Ethnology...
Network,
Artificial intelligence,
Distributed systems...
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
8

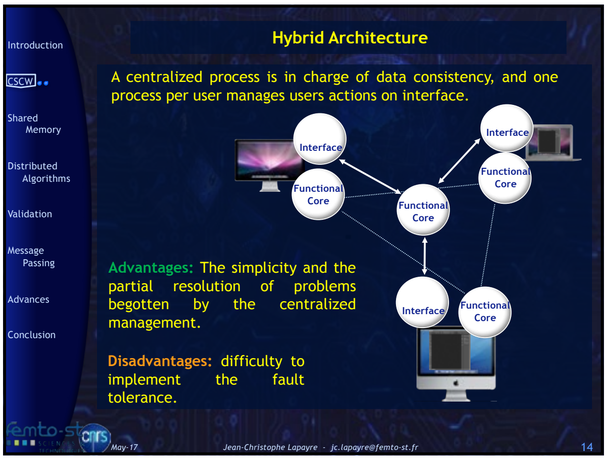

Other point of view
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
AWARENESS
In our team we think that awareness is needed
Are we wrong ? or not ?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
9

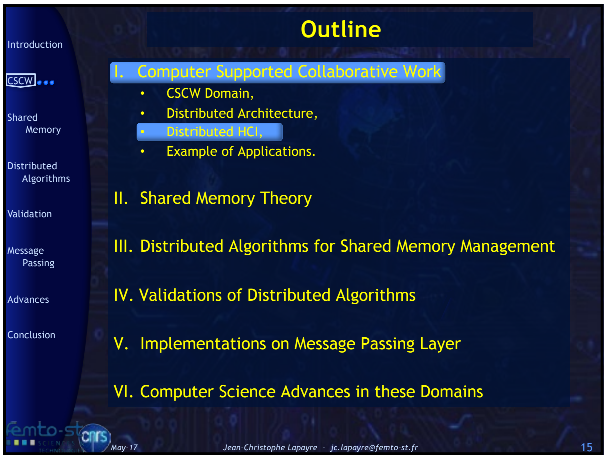

Outline
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
I. Computer Supported Collaborative Work
• CSCW Domain,
• Distributed Architecture,
• Distributed HCI,
• Example of Applications.
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
10

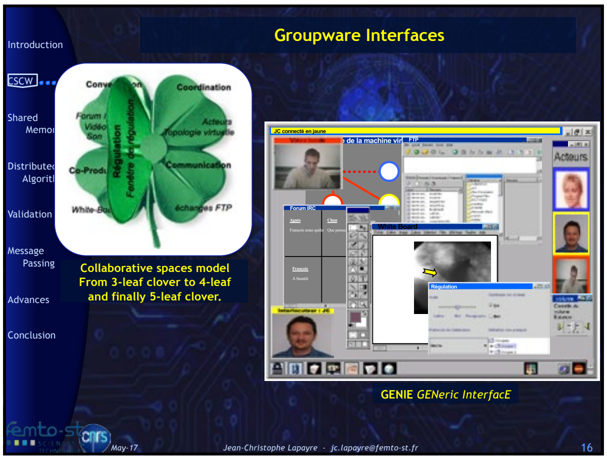

Collaborative Architecture Models
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Centralized
Hybrid
Replicated
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
11

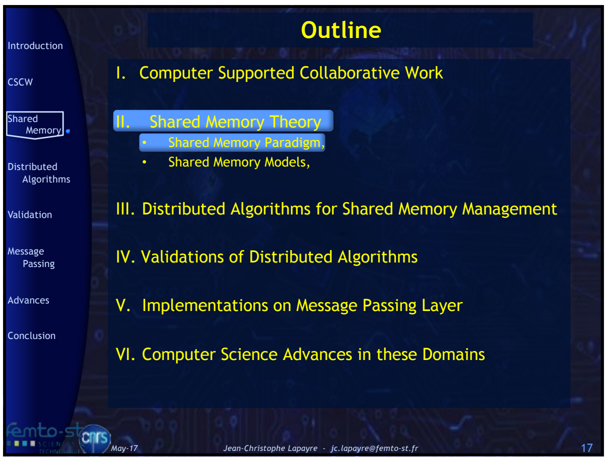

Centralized Architecture
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Only one process manages the consistency of data and actions in the globality of the system.
Advantages: simplicity of implementation for the synchronization and concurrency problems.
Disadvantages: the response time is increased, and fault tolerance can not be processed.
functional core interface
Centralized
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
12

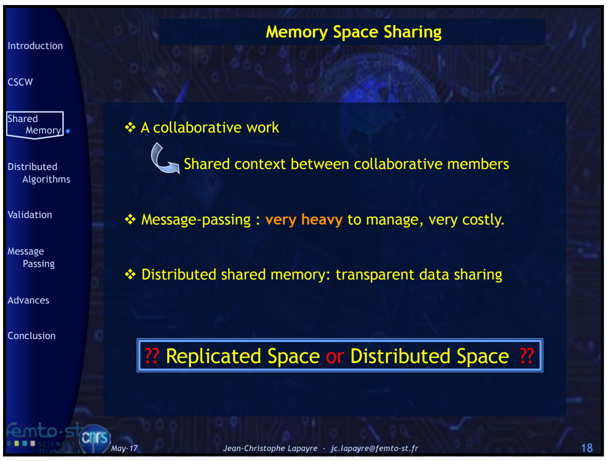

Replicated Architecture
One process is associated to each user. Data are replicated on each site (processor) and the consistency will be maintained using communications between sites.
functional core Interface
functional core Interface
functional core Interface
Advantages: speed due to local accesses, and fault tolerance due to the data redundancy involved by replications.
Disadvantages: difficulty of implementation including data consistency and scheduling of actions.
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
13

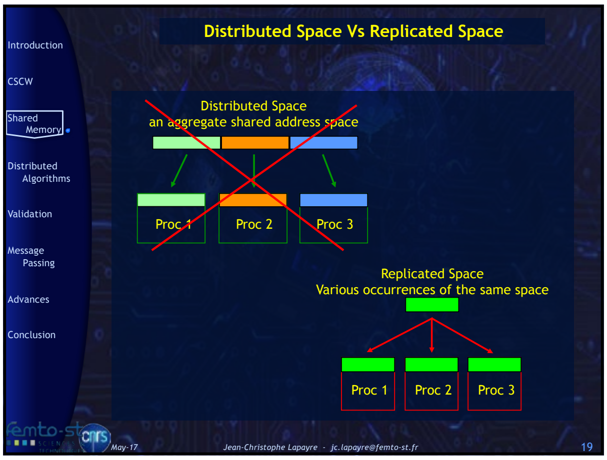

Hybrid Architecture
A centralized process is in charge of data consistency, and one process per user manages users actions on interface.
Interface
Functional Core
Functional Core
Interface
Functional Core
Interface
Functional Core
Advantages: The simplicity and the partial resolution of problems begotten by the centralized management.
Disadvantages: difficulty to implement the fault tolerance.
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
14

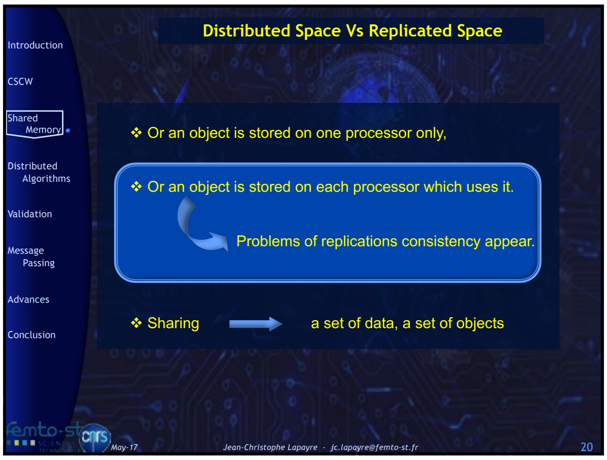

Outline
I. Computer Supported Collaborative Work
CSCW Domain,
Distributed Architecture,
Distributed HCI,
Example of Applications.
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
15

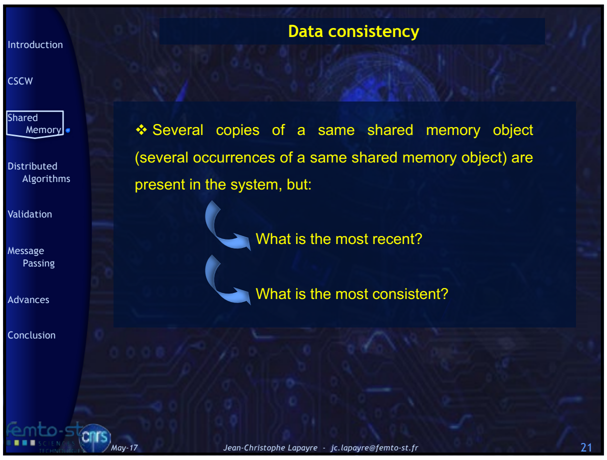

Groupware Interfaces
Coordination
Acteurs
Topologie virtuelle
Communication
échanges FTP
Co-Produ
Régulation
Collaborative spaces model
From 3-leaf clover to 4-leaf and finally 5-leaf clover.
Acteurs
GENIE GENeric InterfacE
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
16

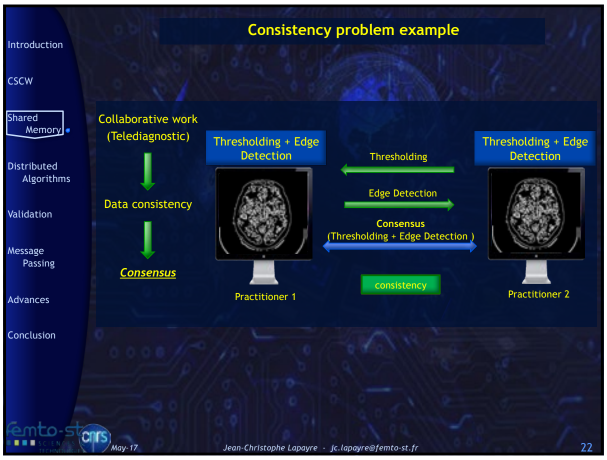

Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
Shared Memory Paradigm,
Shared Memory Models,
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
17

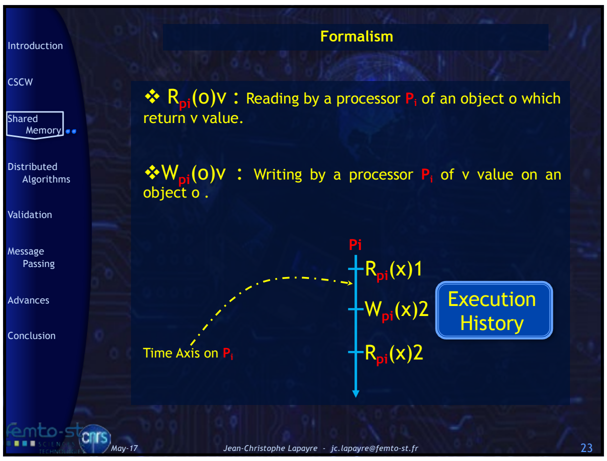

Memory Space Sharing
A collaborative work
Shared context between collaborative members
Message-passing : very heavy to manage, very costly.
Distributed shared memory: transparent data sharing
?? Replicated Space or Distributed Space ??
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
18

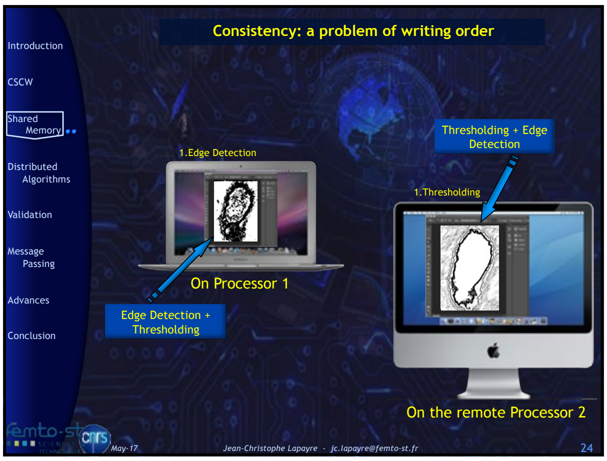

Distributed Space Vs Replicated Space
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Distributed Space
an aggregate shared address space
Proc 1
Proc 2
Proc 3
Replicated Space
Various occurrences of the same space
Proc 1
Proc 2
Proc 3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
19


Distributed Space Vs Replicated Space
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❖ Or an object is stored on one processor only,
❖ Or an object is stored on each processor which uses it.
Problems of replications consistency appear.
❖ Sharing
a set of data, a set of objects
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
20


Data consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❖ Several copies of a same shared memory object (several occurrences of a same shared memory object) are present in the system, but:
What is the most recent?
What is the most consistent?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
21


Consistency problem example
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Collaborative work (Telediagnostic)
Data consistency
Consensus
Thresholding + Edge Detection
Thresholding
Edge Detection
Consensus (Thresholding + Edge Detection)
consistency
Thresholding + Edge Detection
Practitioner 1
Practitioner 2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
22


Formalism
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❖ R_{pi}(o)v : Reading by a processor P_i of an object o which return v value.
❖W_{pi}(o)v : Writing by a processor P_i of v value on an object o .
Pi
R_{pi}(x)1
W_{pi}(x)2
R_{pi}(x)2
Execution History
Time Axis on P_i
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
23

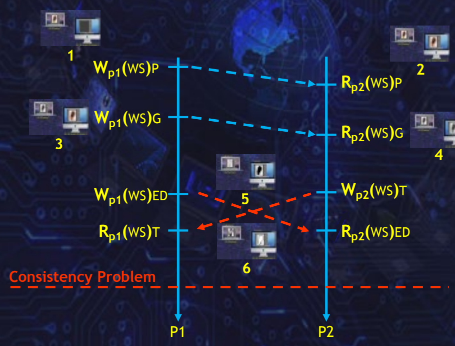

Consistency: a problem of writing order
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
1.Edge Detection
On Processor 1
Edge Detection + Thresholding
Thresholding + Edge Detection
1.Thresholding
On the remote Processor 2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
24

Now, your Job: to write this execution history

- $R_{pi}(o)v$: Reading by a processor p of an object o which return v value.
- $W_{pi}(o)v$: Writing by a processor p of v value on an object o .
- Writing sequences in our example :
  - W_{p1}(WorkSpace)Picture noted W_{p1}(WS)P,
  - W_{p1}(WorkSpace)Gray Levels Transformation noted W_{p1}(WP)G,
  - W_{p1}(WorkSpace)Edge Detection noted W_{p1}(WS)ED,
  - W_{p2}(WorkSpace)Thresholding noted W_{p2}(WS)T.

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

Correction
Execution History: Scheduling Problem



Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

Outline

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

Consistency types

- Non-synchronized
  - 5 types,
  - 3 types really in used in applications,
  - 2 theoretical types.
- Synchronized

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

Definition using constraints

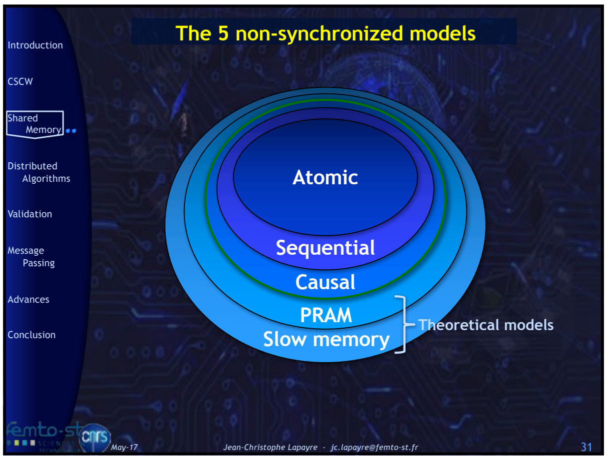
- Some shared memory need less constraints, to have a management more flexible.
- Weaken the consistency allows better performance in execution .
- The different types of consistency are classified following to the constraints: time, order...

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

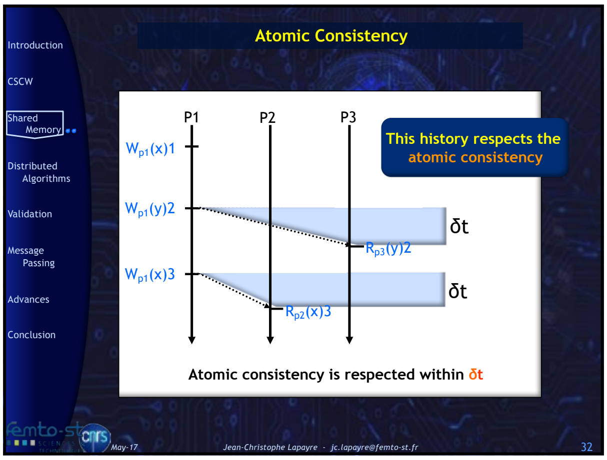
The Constraints

- C1: The operations are observable simultaneously on all processors.
- C2: The global observed order of the operations is the same on each processor. All processors observe the same sequence but not on the same time.
- C3: The only one execution order, which is respected, is the causality (read / write).
- C4: The execution order is only respected for operations made on the same processor.
- C5: The execution order is only respected for operations made on the same processor and on the same object.

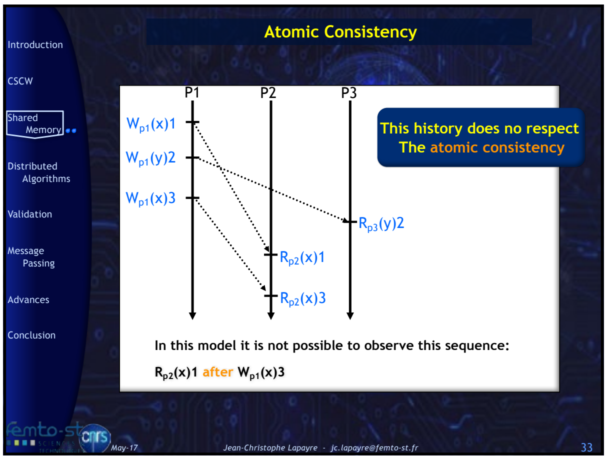
In these models all operations can be oberved on all processors



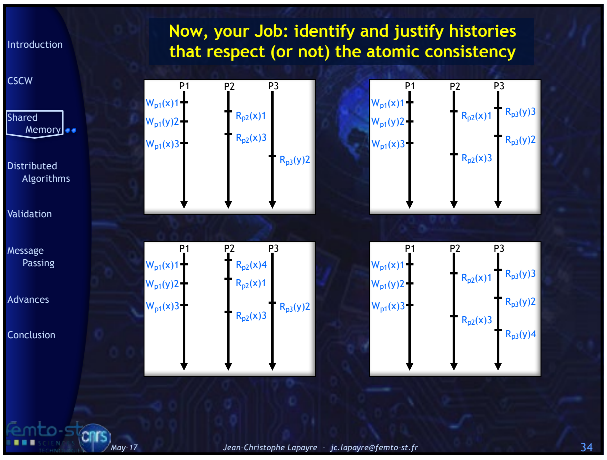
The 5 non-synchronized models
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Atomic
Sequential
Causal
PRAM
Slow memory
Theoretical models
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
31



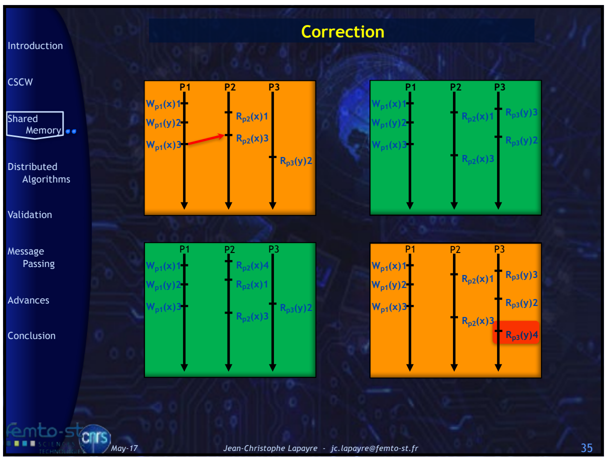
Atomic Consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
P1
P2
P3
This history respects the atomic consistency
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp3(y)2
Rp2(x)3
δt
δt
Atomic consistency is respected within δt
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
32



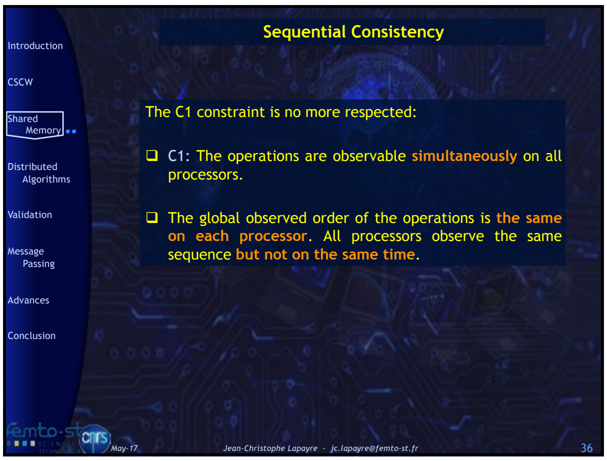
Atomic Consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
P1
P2
P3
This history does no respect The atomic consistency
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp3(y)2
Rp2(x)1
Rp2(x)3
In this model it is not possible to observe this sequence:
Rp2(x)1 after Wp1(x)3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
33



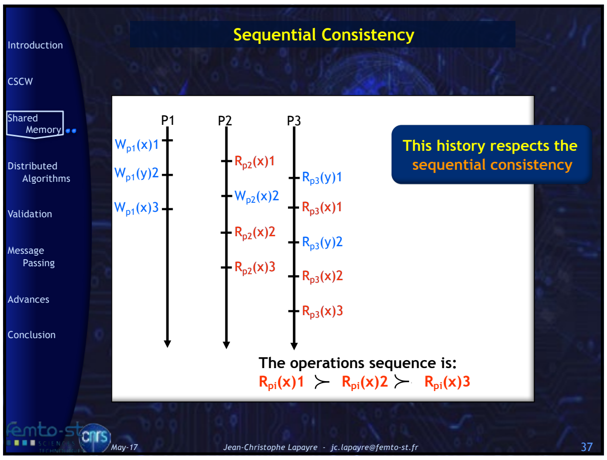
Now, your Job: identify and justify histories that respect (or not) the atomic consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
34

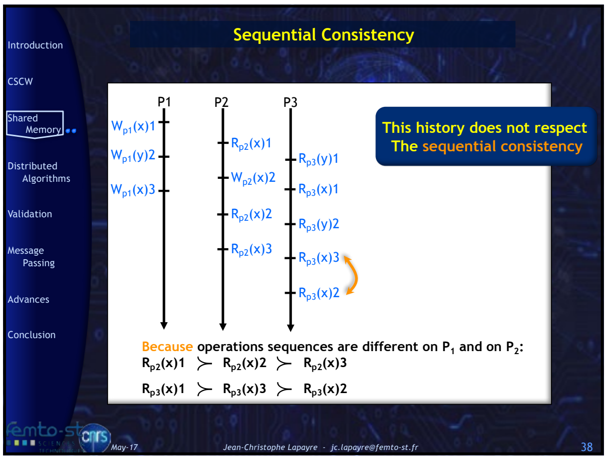


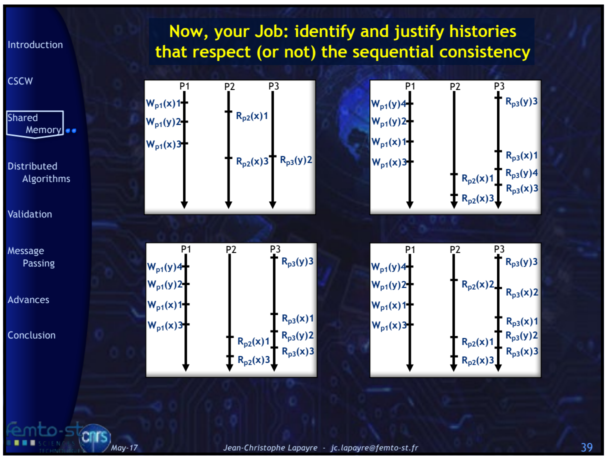
Correction
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
35

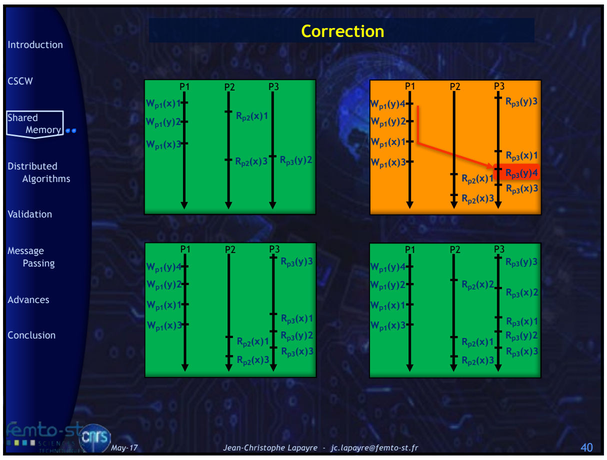


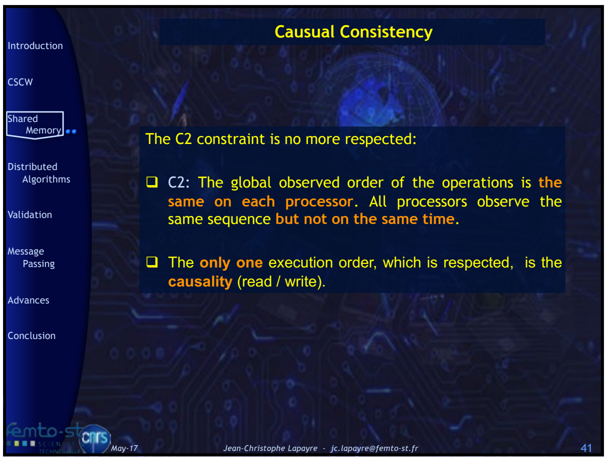
Sequential Consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
The C1 constraint is no more respected:
C1: The operations are observable simultaneously on all processors.
The global observed order of the operations is the same on each processor. All processors observe the same sequence but not on the same time.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
36

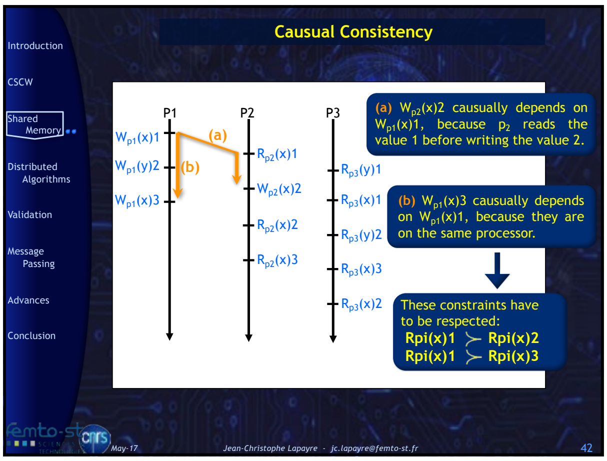

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Sequential Consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)2
Rp3(x)3
This history respects the sequential consistency
The operations sequence is:
Rpi(x)1 ≻ Rpi(x)2 ≻ Rpi(x)3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
37

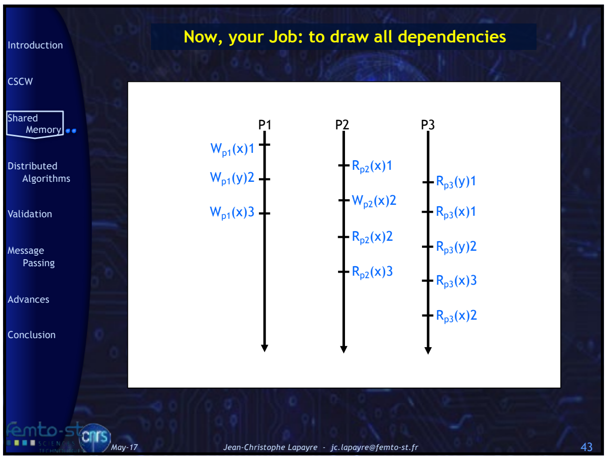

Sequential Consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp3(x)2
This history does not respect The sequential consistency
Because operations sequences are different on P1 and on P2:
Rp2(x)1 ≻ Rp2(x)2 ≻ Rp2(x)3
Rp3(x)1 ≻ Rp3(x)3 ≻ Rp3(x)2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
38

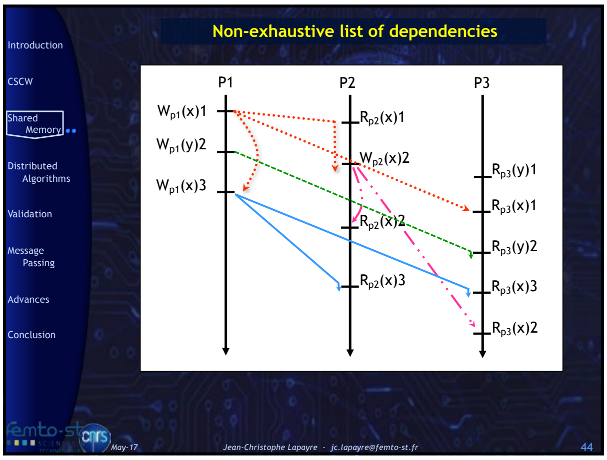

Now, your Job: identify and justify histories that respect (or not) the sequential consistency
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
39

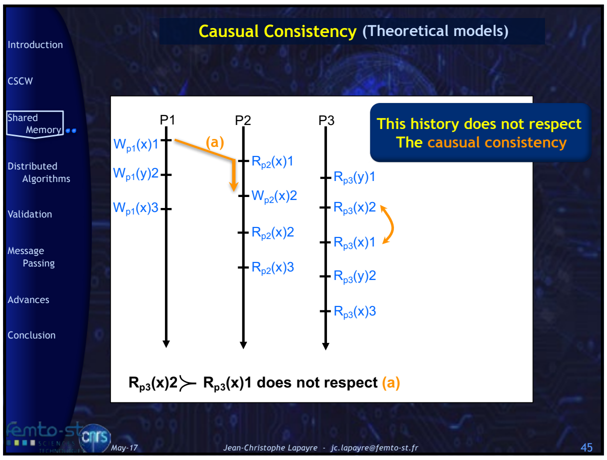

Correction
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
40

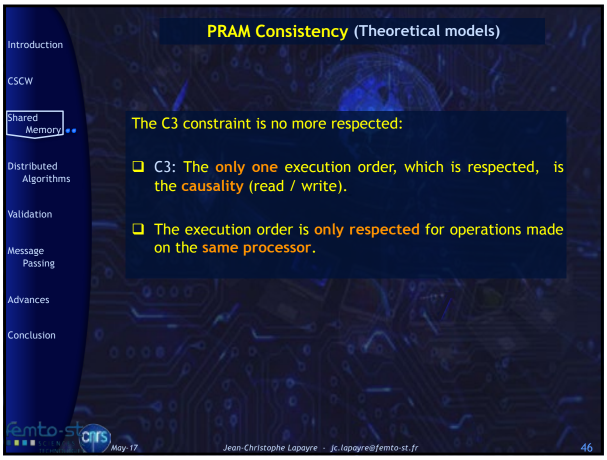

Causual Consistency
The C2 constraint is no more respected:
C2: The global observed order of the operations is the same on each processor. All processors observe the same sequence but not on the same time.
The only one execution order, which is respected, is the causality (read / write).
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
41

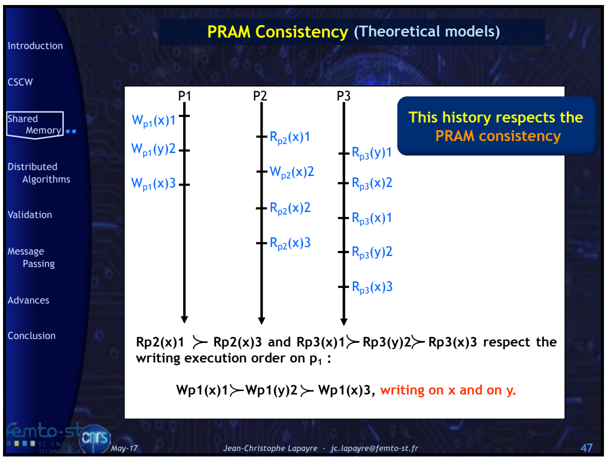

Causual Consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp3(x)2
(a) Wp2(x)2 causually depends on Wp1(x)1, because p2 reads the value 1 before writing the value 2.
(b) Wp1(x)3 causually depends on Wp1(x)1, because they are on the same processor.
These constraints have to be respected:
Rpi(x)1 ≻ Rpi(x)2
Rpi(x)1 ≻ Rpi(x)3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
42

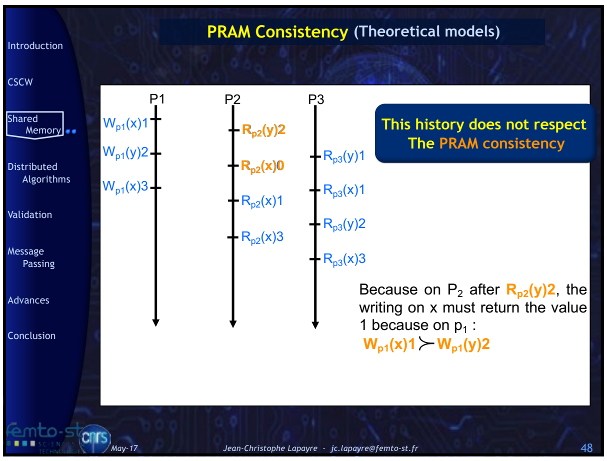

Now, your Job: to draw all dependencies
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp3(x)2
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
43


Non-exhaustive list of dependencies
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp3(x)2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
44


Causal Consistency (Theoretical models)
This history does not respect
The causual consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
(a)
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)2
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp3(x)2 ≻ Rp3(x)1 does not respect (a)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
45


PRAM Consistency (Theoretical models)
The C3 constraint is no more respected:
❑ C3: The only one execution order, which is respected, is the causality (read / write).
❑ The execution order is only respected for operations made on the same processor.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
46


PRAM Consistency (Theoretical models)
This history respects the
PRAM consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(x)1
Wp2(x)2
Rp2(x)2
Rp2(x)3
Rp3(y)1
Rp3(x)2
Rp3(x)1
Rp3(y)2
Rp3(x)3
Rp2(x)1 ≻ Rp2(x)3 and Rp3(x)1≻Rp3(y)2≻Rp3(x)3 respect the writing execution order on p1 :
Wp1(x)1≻Wp1(y)2≻Wp1(x)3, writing on x and on y.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
47


PRAM Consistency (Theoretical models)
This history does not respect
The PRAM consistency
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rp2(y)2
Rp2(x)0
Rp2(x)1
Rp2(x)3
Rp3(y)1
Rp3(x)1
Rp3(y)2
Rp3(x)3
Because on P2 after Rp2(y)2, the writing on x must return the value 1 because on p1 :
Wp1(x)1≻Wp1(y)2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
48

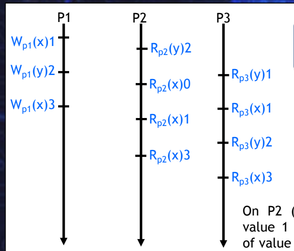
Slow Memory (Theoretical models)

The C4 constraint is no more respected:

- C4: The execution order is only respected for operations made on the same processor.
- The execution order is only respected for operations made on the same processor and on the same object.

Slow Memory Consistency

Introduction CSCW Shared Memory Distributed Algorithms Validation Message Passing Advances Conclusion

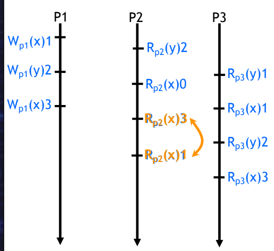


This history respects the Slow memory consistency

On P2 (as on P3) the reading of value 1 on x precedes the reading of value 3 on x, so the P1's writings on x are respected by these readings:
$W_{p1}(x)1 \succ W_{p1}(x)3$

Slow Memory Consistency

Introduction CSCW Shared Memory Distributed Algorithms Validation Message Passing Advances Conclusion



This history does not respect The slow memory consistency

The P1's writings on x:
$W_{p1}(x)1 \succ W_{p1}(x)3$

are not respected on P_2

Synchronized Consistency Models

Introduction CSCW Shared Memory Distributed Algorithms Validation Message Passing Advances Conclusion

In these first five models, all writing operations have to be observable by all processors.

- In these last two models, all operations are not necessary observable. Only operations, which are synchronized, are observable.
- Two models are exposed:
  - The weak ordering,
  - The release consistency.

Session 2

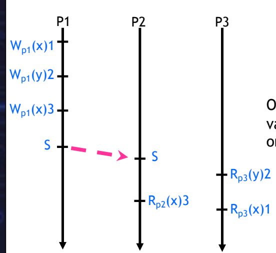
Weak Ordering

Introduction CSCW Shared Memory Distributed Algorithms Validation Message Passing Advances Conclusion

- A new operator is defined S, used with $R_{pi}(o)v$ et $W_{pi}(o)v$.
- Remark: it is possible to defined S for each object:
  S_x, S_y

Weak Ordering

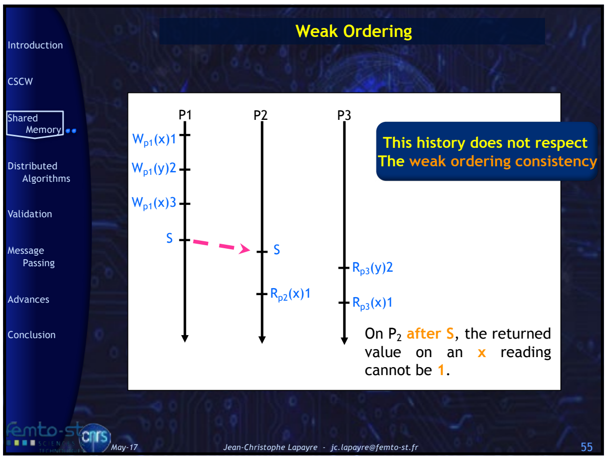
Introduction CSCW Shared Memory Distributed Algorithms Validation Message Passing Advances Conclusion

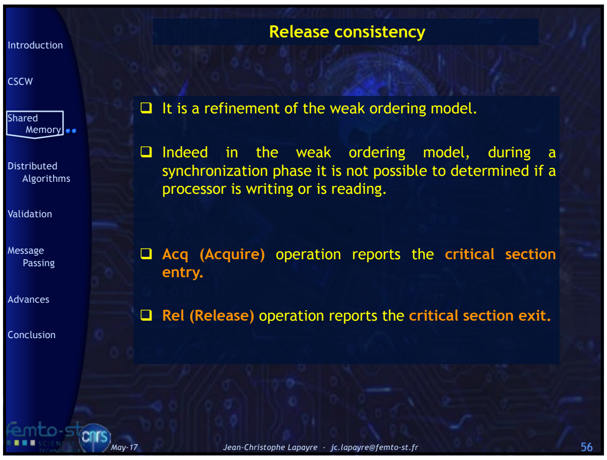


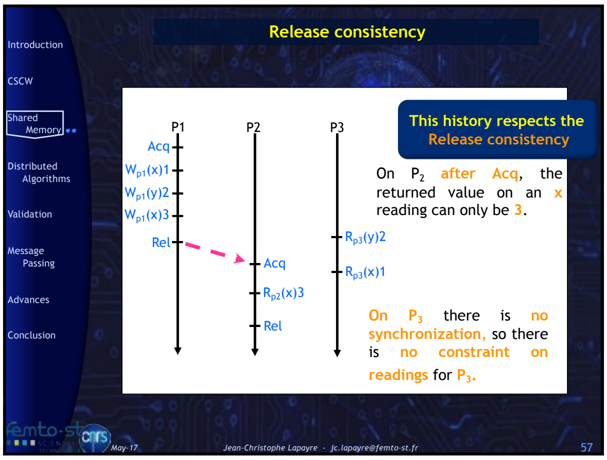
This history respects the weak ordering consistency

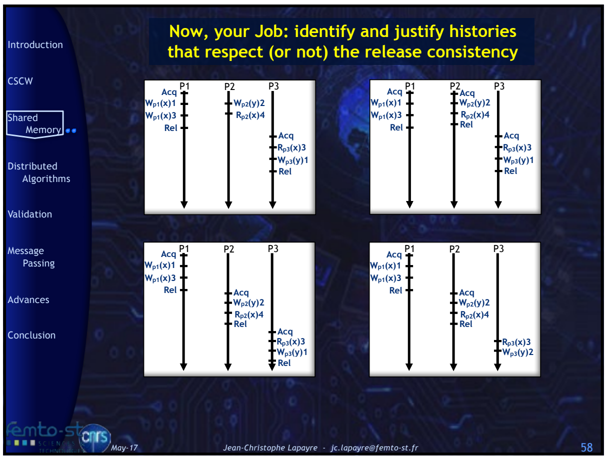
On P_2 after S, the returned value on an x reading can only be 3.

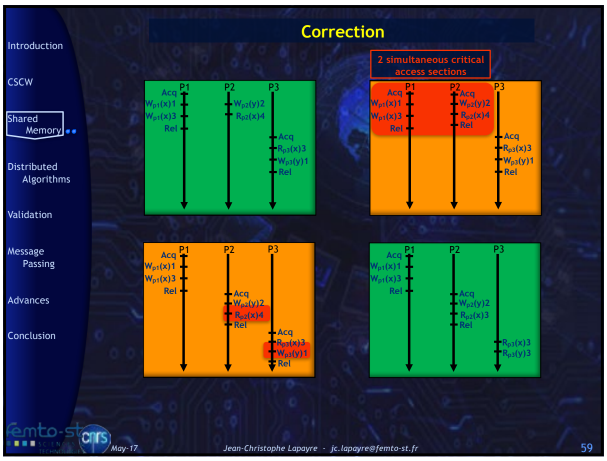
On P_3 there is no synchronization, so there is no constraint on readings for P_3.

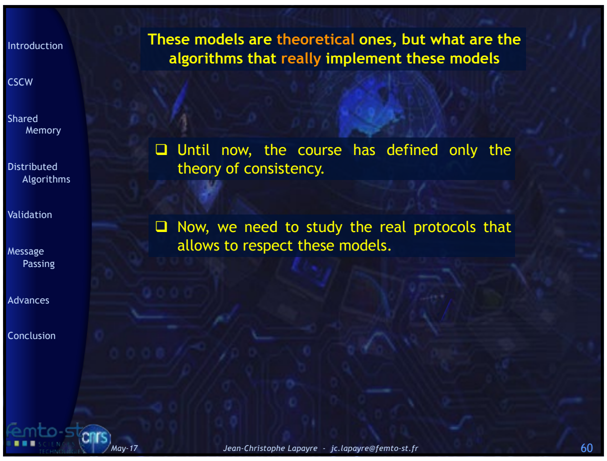

Weak Ordering
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
P1
P2
P3
Wp1(x)1
Wp1(y)2
Wp1(x)3
S
S
Rp3(y)2
Rp2(x)1
Rp3(x)1
This history does not respect
The weak ordering consistency
On P2 after S, the returned value on an x reading cannot be 1.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
55


Release consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
It is a refinement of the weak ordering model.
Indeed in the weak ordering model, during a synchronization phase it is not possible to determined if a processor is writing or is reading.
Acq (Acquire) operation reports the critical section entry.
Rel (Release) operation reports the critical section exit.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
56


Release consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
P1
P2
P3
Acq
Wp1(x)1
Wp1(y)2
Wp1(x)3
Rel
Acq
Rp2(x)3
Rel
Rp3(y)2
Rp3(x)1
This history respects the Release consistency
On P2 after Acq, the returned value on an x reading can only be 3.
On P3 there is no synchronization, so there is no constraint on readings for P3.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
57


Now, your Job: identify and justify histories that respect (or not) the release consistency
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
58


Correction
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
2 simultaneous critical access sections
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
59


These models are theoretical ones, but what are the algorithms that really implement these models
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Until now, the course has defined only the theory of consistency.
Now, we need to study the real protocols that allows to respect these models.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
60

Outline

Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation · Message Passing · Advances · Conclusion

I. Computer Supported Collaborative Work

II. Shared Memory Theory

III. Distributed Algorithms for Shared Memory Management
- Invalidation Protocol,
- Pilgrim Protocol,
- Shared Memory guarantied Models.

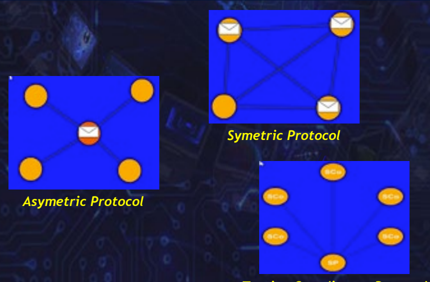
IV. Validations of Distributed Algorithms

V. Implementations on Message Passing Layer

VI. Computer Science Advances in these Domains

Broadcast of Information

- To guarantee that messages arrive ordered, without loss and duplication
- To facilitate the management of shared data consistency
- 3 types of protocols exist:



Outline

I. Computer Supported Collaborative Work

II. Shared Memory Theory

III. Distributed Algorithms for Shared Memory Management
- Invalidation Protocol,
- Pilgrim Protocol,
- Shared Memory guarantied Models.

IV. Validations of Distributed Algorithms

V. Implementations on Message Passing Layer

VI. Computer Science Advances in these Domains

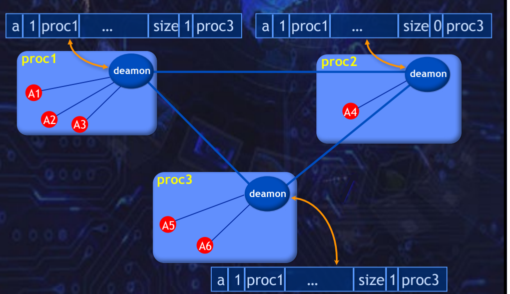
Invalidation Protocol Specifications

- Replicated memory: an object is stored on each processor which uses it.
- The grain of the application is the object.
- This protocol uses the technics of invalidation on writing.
- From an object point of view, this protocol is a simpleWriter/multipleReaders one.
- From the memory point of view, this protocol is a multipleWriters/multipleReaders one.
- The owner of an object is the only one which can write on this object

After each writing an invalidation must be broadcast → **For each object**

Invalidation Protocol

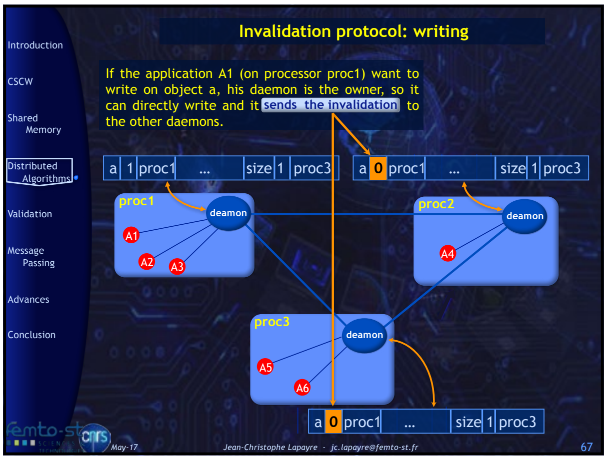
- This implementation was made using a message passing platform. On each server a deamon is running.

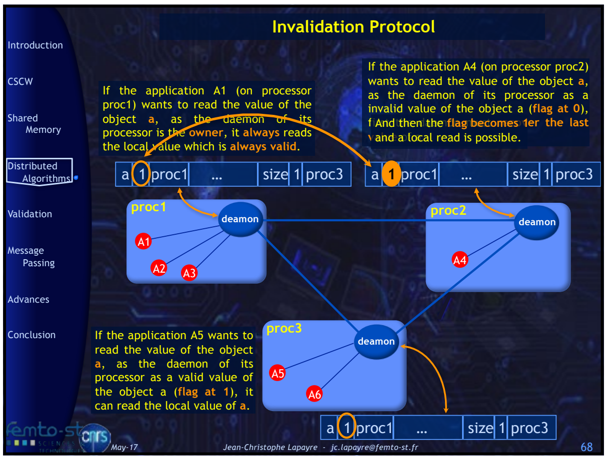


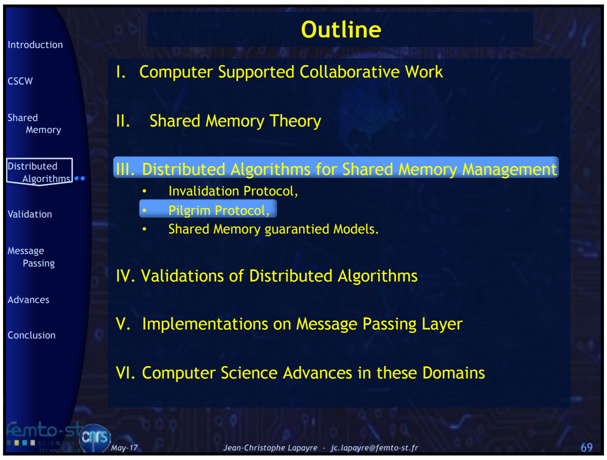
Invalidation Protocol

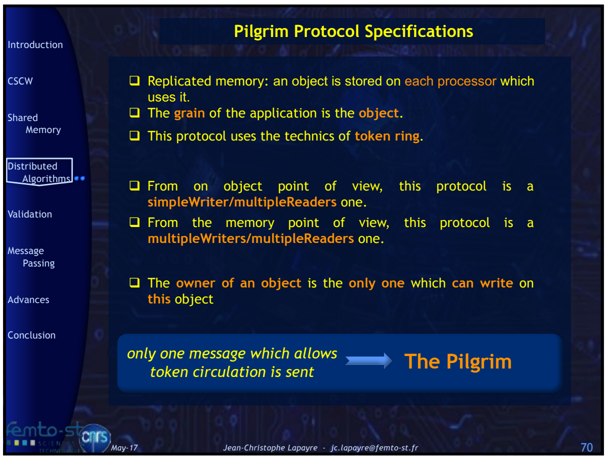
Writer
- The owner
  - writing,
  - Then, invalidation broadcasting.
- The non-owners
  - request to recover the ownership,
  - Then, ownership change
    - Writing
    - Then, invalidation broadcasting.

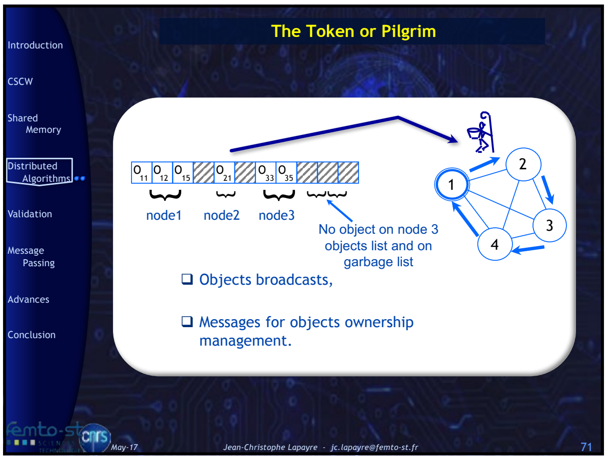
Reader
- The owner
  - Local reading.
- The non-owners
  - Local reading if valid, if not recover remote value (in the same time the local flag value becomes 1)

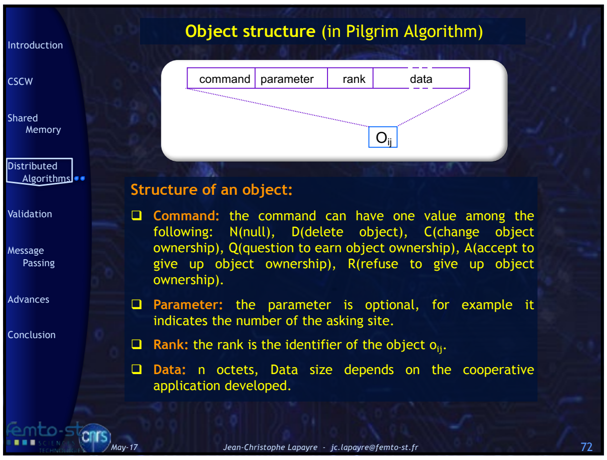

Invalidation protocol: writing
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
If the application A1 (on processor proc1) want to write on object a, his daemon is the owner, so it can directly write and it sends the invalidation to the other daemons.
a 1 proc1 ... size 1 proc3
a 0 proc1 ... size 1 proc3
proc1
deamon
A1
A2
A3
proc2
deamon
A4
proc3
deamon
A5
A6
a 0 proc1 ... size 1 proc3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
67


Invalidation Protocol
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
If the application A1 (on processor proc1) wants to read the value of the object a, as the daemon of its processor is the owner, it always reads the local value which is always valid.
If the application A4 (on processor proc2) wants to read the value of the object a, as the daemon of its processor as a invalid value of the object a (flag at 0), And then the flag becomes 1er the last and a local read is possible.
a 1 proc1 ... size 1 proc3
a 1 proc1 ... size 1 proc3
proc1
deamon
A1
A2
A3
proc2
deamon
A4
proc3
deamon
A5
A6
If the application A5 wants to read the value of the object a, as the daemon of its processor as a valid value of the object a (flag at 1), it can read the local value of a.
a 1 proc1 ... size 1 proc3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
68


Outline
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
• Invalidation Protocol,
• Pilgrim Protocol,
• Shared Memory guarantied Models.
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
69


Pilgrim Protocol Specifications
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Replicated memory: an object is stored on each processor which uses it.
The grain of the application is the object.
This protocol uses the technics of token ring.
From on object point of view, this protocol is a simpleWriter/multipleReaders one.
From the memory point of view, this protocol is a multipleWriters/multipleReaders one.
The owner of an object is the only one which can write on this object
only one message which allows token circulation is sent
The Pilgrim
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
70


The Token or Pilgrim
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
node1
node2
node3
No object on node 3 objects list and on garbage list
1
2
3
4
Objects broadcasts,
Messages for objects ownership management.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
71


Object structure (in Pilgrim Algorithm)
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
command
parameter
rank
data
Oij
Structure of an object:
Command: the command can have one value among the following: N(null), D(delete object), C(change object ownership), Q(question to earn object ownership), A(accept to give up object ownership), R(refuse to give up object ownership).
Parameter: the parameter is optional, for example it indicates the number of the asking site.
Rank: the rank is the identifier of the object oij.
Data: n octets, Data size depends on the cooperative application developed.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
72

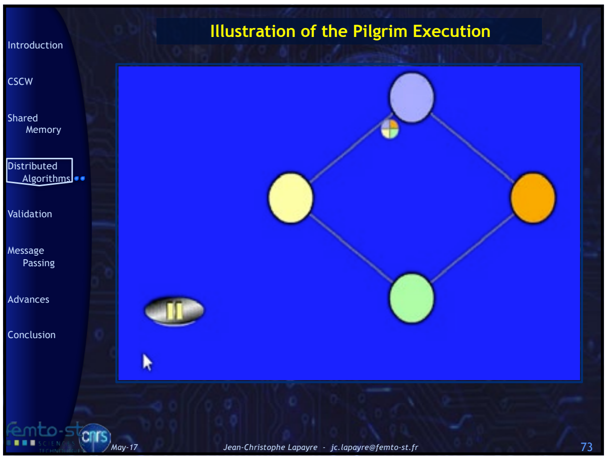
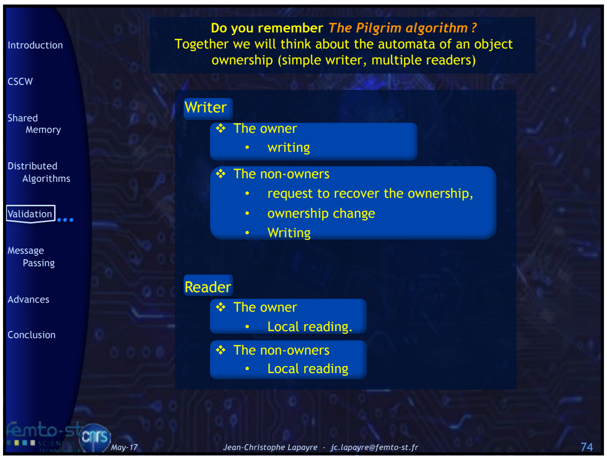
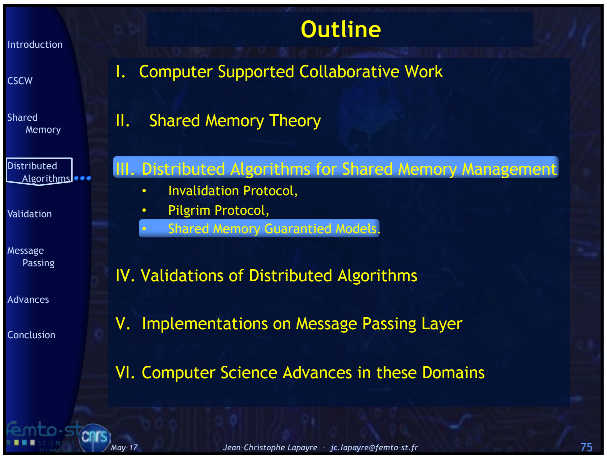
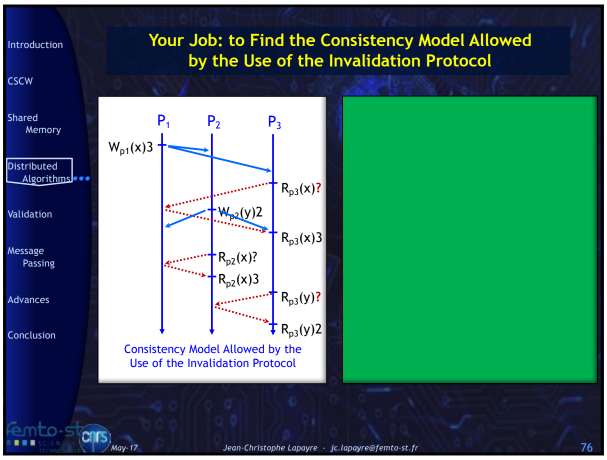
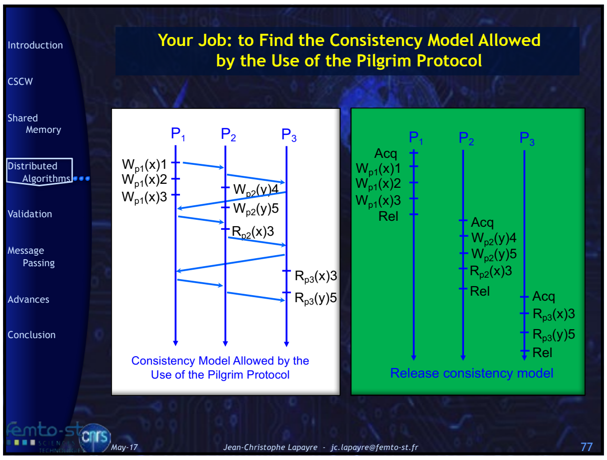


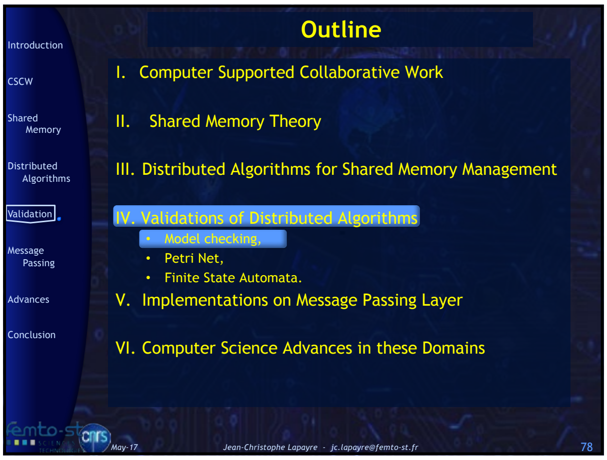
Illustration of the Pilgrim Execution
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
73


Do you remember The Pilgrim algorithm ?
Together we will think about the automata of an object ownership (simple writer, multiple readers)
Writer
The owner
writing
The non-owners
request to recover the ownership,
ownership change
Writing
Reader
The owner
Local reading.
The non-owners
Local reading
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
74


Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
Invalidation Protocol,
Pilgrim Protocol,
Shared Memory Guarantied Models.
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
75


Your Job: to Find the Consistency Model Allowed by the Use of the Invalidation Protocol
P_1 P_2 P_3
$W_{p1}(x)3$
$R_{p3}(x)?$
$W_{p2}(y)2$
$R_{p3}(x)3$
$R_{p2}(x)?$
$R_{p2}(x)3$
$R_{p3}(y)?$
$R_{p3}(y)2$
Consistency Model Allowed by the Use of the Invalidation Protocol
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
76


Your Job: to Find the Consistency Model Allowed by the Use of the Pilgrim Protocol
P_1 P_2 P_3
$W_{p1}(x)1$
$W_{p1}(x)2$
$W_{p1}(x)3$
$W_{p2}(y)4$
$W_{p2}(y)5$
$R_{p2}(x)3$
$R_{p3}(x)3$
$R_{p3}(y)5$
Consistency Model Allowed by the Use of the Pilgrim Protocol
Acq
Rel
Release consistency model
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
77


Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
Model checking,
Petri Net,
Finite State Automata.
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
78

Validation with model checking tools (average 40 existing MCTs)
The most famous SPIN

SPIN overview

- open-source software tool (http://spinroot.com),
- (freely available since 1991),
- *one of the most prominent tools for formal verification of distributed software systems,*
- developed by Gerald Holzmann at Bell Labs (beginning in 1980),
- awarded the prestigious System Software Award 2001 by the ACM,
- Primer and Reference Manual [Hol03].

To use SPIN, you need to implement your distributed protocol in ProMeLa Language.

ProMeLa (Process Meta Language)
→ the name SPIN stands for *Simple ProMeLa Interpreter*

SPIN

- Channel declarations:
  chan ChannelName = [capacity] of {mtype, T_0,T_2, . . . ,T_{k-1}};

- ChannelName : name of the channel,
- capacity : capacity of the FIFO channel (O in Pilgrim),
- T_i, $0 \leq i \leq k-1$: type of transmittable data (tuples),
  - *Example*
    {mtype,byte,byte,byte,int,byte,byte,byte,int,byte,byte,byte,int};

- Mtype is the Label: *example mtype = {pilgrim,ackn,premier}*

Communication:
- synchronous message passing ← capacity 0,
- asynchronous message passing ← capacity > 1,

Communication actions:
- sending: ChannelName! $expr_1$, $expr_2$, . . . , $expr_k$;
  - Example: *token[node1]!pilgrim(c1,f1,p1,d1,c2,f2,p2,d2,c3,f3,p3,d3);*
- receiving: ChannelName? x_1, x_2, . . . , x_k ;
  - Example: *ack[node1]?ackn*

SPIN

- *Repetitive command/loop:*

```
do
   :: (g) -> statement               ← WHILE (g) DO statement OD
od
```

- *Conditional command:*

```
if
   :: (g) -> statement1              ← IF (g) THEN statement1 ELSE statement 2 FI
   :: ( ¬g) -> statement2
fi
```

- *Non deterministic loop:*

```
do
   :(true) -> statement1
   :(true) -> statement2
od
```


SPIN: Classic Presented Case

- If several parts are kept activable simultaneously when each of them has an *equal probability chance* being selected: *non deterministic decisions*. This means that during checking *all solutions will be verified.*

```
proctype transmitter(){
            byte info = 1
            do
                  :: (true) ->support! data,info;
                  :: (true) ->support! control,info;
             od
}
```

```
proctype receiver(){
      byte info;
      do      /* loop => exit with break */
        :: (true) -> support? data,info ->printf("received data message: %d", info)
        :: (true) -> support? control,info -> printf("received control message: %d", info)
      od
}
```

```
init {
         atomic {      /* execution of an indivisible sequence */
                       /* Parallel processes instantiation      */

              run transmitter();
              run receiver();
         }
}
```

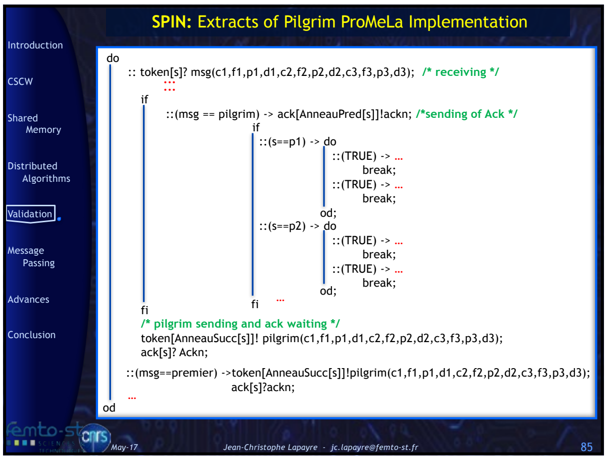

SPIN: Extracts of Pilgrim ProMeLa Implementation

```
#define N 100
#define R 102
#define A 103
#define Q 104

#define TRUE 1
#define FALSE 0

#define PROCESSEUR 3
```

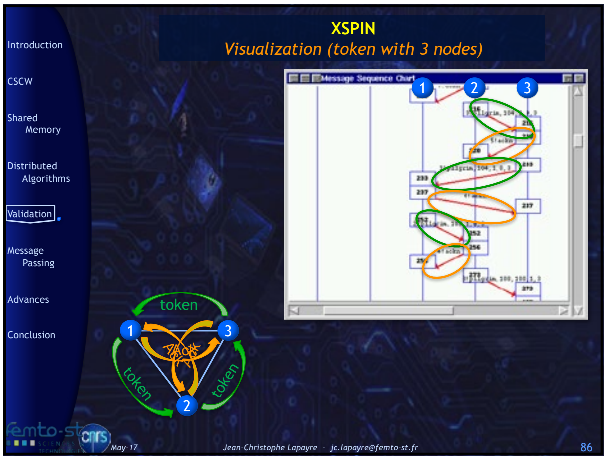
```
mtype = {pilgrim,ackn,premier}

chan token[PROCESSEUR ] = [0] of
{mtype,byte,byte,byte,int,byte,byte,byte,int,byte,byte,byte,int};

chan ack[PROCESSEUR ] = [0] of {mtype};
```


SPIN: Extracts of Pilgrim ProMeLa Implementation

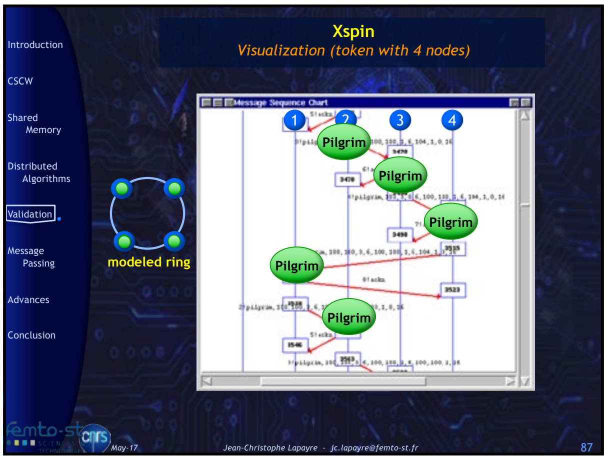
```
Init {
  byte  s;
  atomic  {
        run premierenvoi();
        /* Creation of SUCC and PRED lists */
        s=0;
        do
            :: (s <  PROCESSEUR ) -> AnneauSucc[s] = (s+1) % PROCESSEUR ;
                                     AnneauPred[s] = (PROCESSEUR  + s-1) % PROCESSEUR ;
                                     s=s+1 ;
            :: (s >= PROCESSEUR )-> break;
        od;
       /* Successive startup of the different nodes*/
       s=0;
       do
            :: (s <  PROCESSEUR ) -> run Recevoir(s);
                                     s=s+1;
            :: (s >= PROCESSEUR )-> break;
       od
  }
}
```



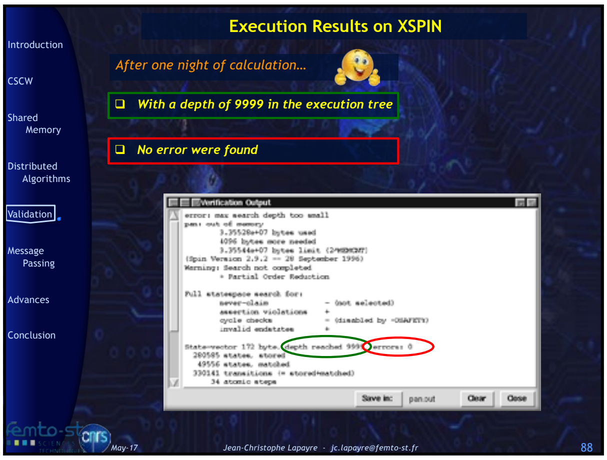
SPIN: Extracts of Pilgrim ProMeLa Implementation
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
do
:: token[s]? msg(c1,f1,p1,d1,c2,f2,p2,d2,c3,f3,p3,d3); /* receiving */
if
::(msg == pilgrim) -> ack[AnneauPred[s]]!ackn; /*sending of Ack */
if
::(s==p1) -> do
::(TRUE) -> ...
break;
::(TRUE) -> ...
break;
od;
::(s==p2) -> do
::(TRUE) -> ...
break;
::(TRUE) -> ...
break;
od;
fi ...
fi
/* pilgrim sending and ack waiting */
token[AnneauSucc[s]]! pilgrim(c1,f1,p1,d1,c2,f2,p2,d2,c3,f3,p3,d3);
ack[s]? Ackn;
::(msg==premier) ->token[AnneauSucc[s]]!pilgrim(c1,f1,p1,d1,c2,f2,p2,d2,c3,f3,p3,d3);
ack[s]?ackn;
...
od
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
85



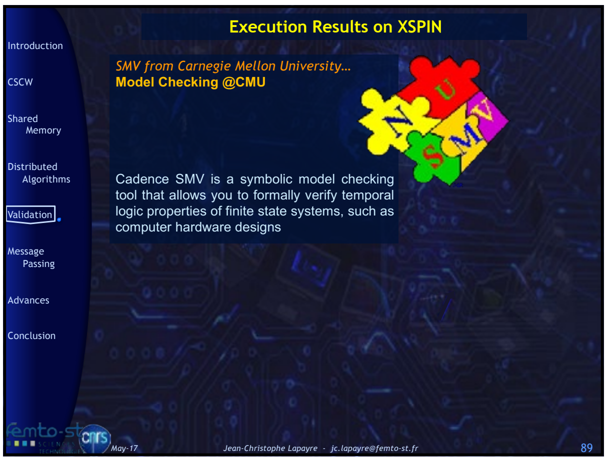
XSPIN
Visualization (token with 3 nodes)
Message Sequence Chart
token
1
2
3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
86



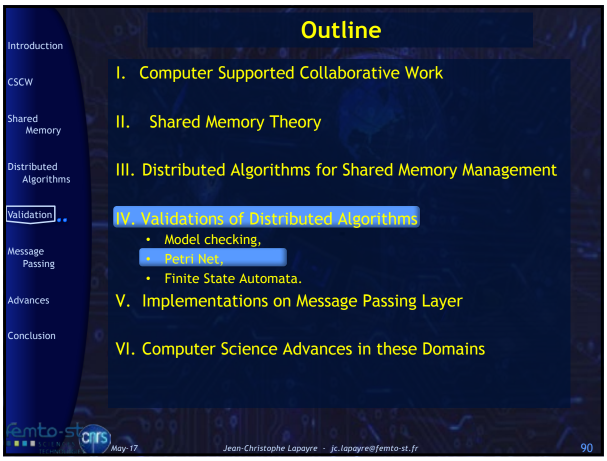
Xspin
Visualization (token with 4 nodes)
Message Sequence Chart
1
2
3
4
Pilgrim
modeled ring
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
87



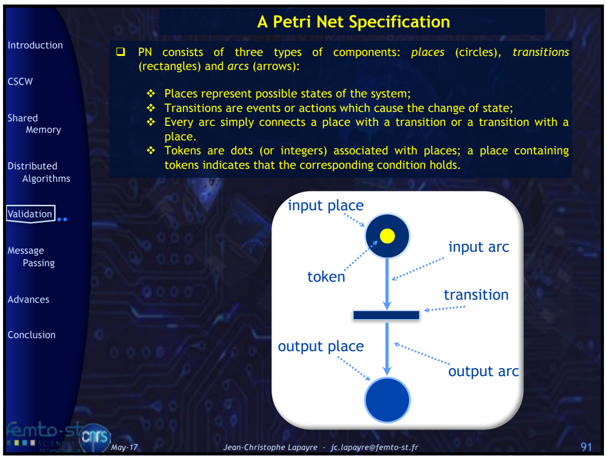
Execution Results on XSPIN
After one night of calculation…
With a depth of 9999 in the execution tree
No error were found
Verification Output
Save in:
Clear
Close
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
88

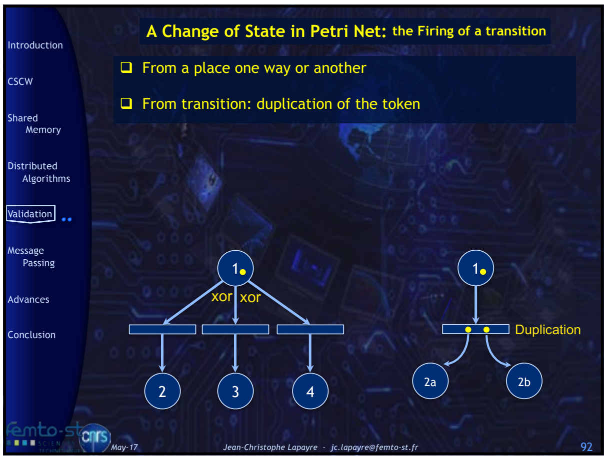


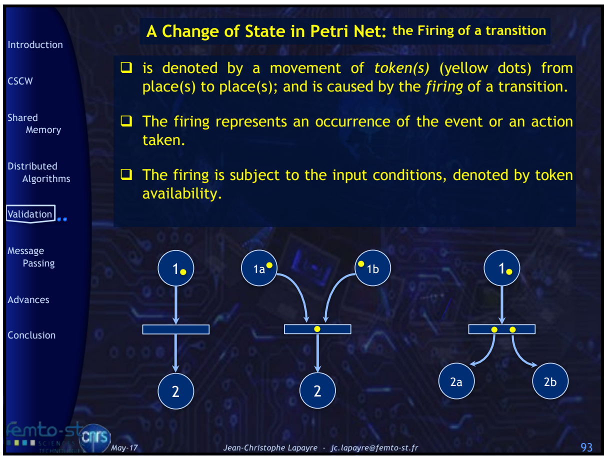
Execution Results on XSPIN
SMV from Carnegie Mellon University…
Model Checking @CMU
Cadence SMV is a symbolic model checking tool that allows you to formally verify temporal logic properties of finite state systems, such as computer hardware designs
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
89

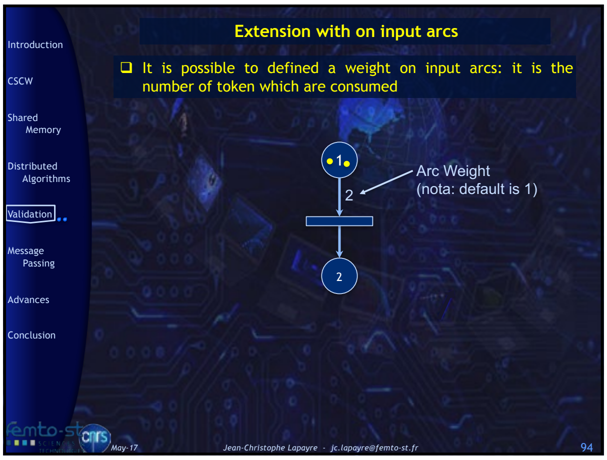


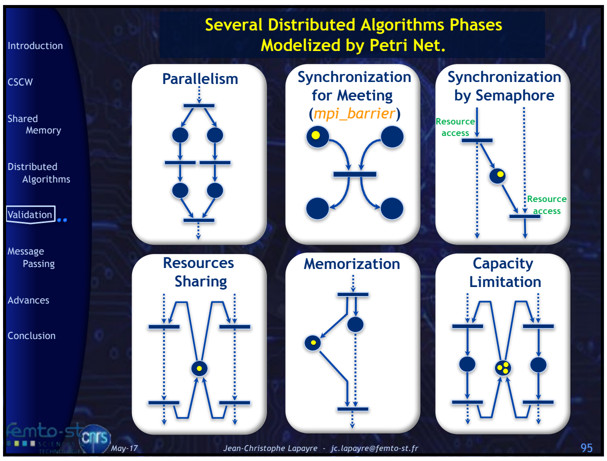
Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
• Model checking,
• Petri Net,
• Finite State Automata.
V. Implementations on Message Passing Layer
VI. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
90

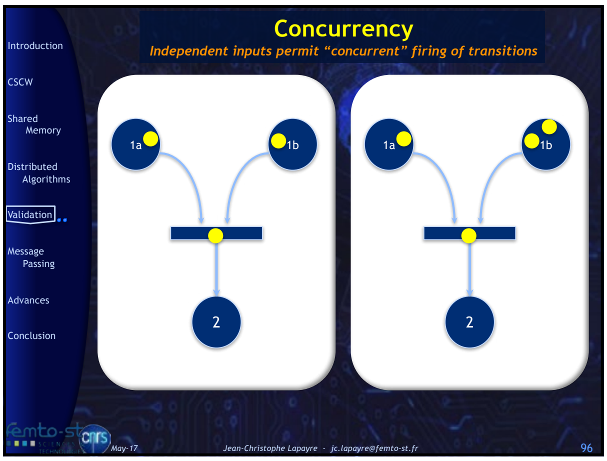

A Petri Net Specification
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
PN consists of three types of components: places (circles), transitions (rectangles) and arcs (arrows):
Places represent possible states of the system;
Transitions are events or actions which cause the change of state;
Every arc simply connects a place with a transition or a transition with a place.
Tokens are dots (or integers) associated with places; a place containing tokens indicates that the corresponding condition holds.
input place
token
input arc
transition
output place
output arc
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
91

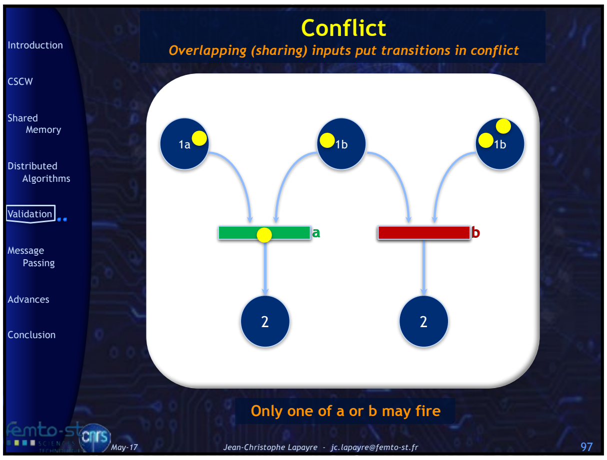

A Change of State in Petri Net: the Firing of a transition
From a place one way or another
From transition: duplication of the token
1
xor
xor
2
3
4
Duplication
2a
2b
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
92

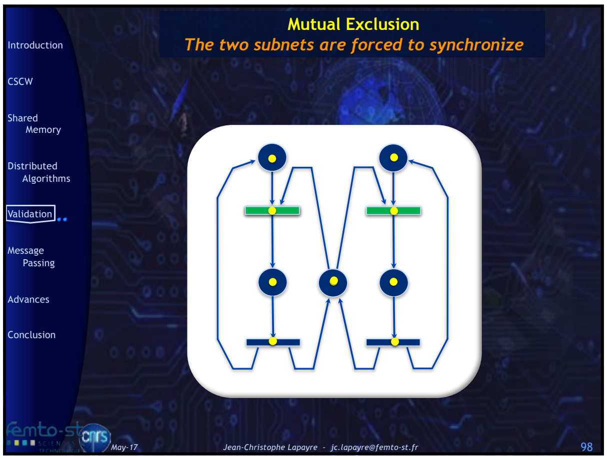

A Change of State in Petri Net: the Firing of a transition
is denoted by a movement of token(s) (yellow dots) from place(s) to place(s); and is caused by the firing of a transition.
The firing represents an occurrence of the event or an action taken.
The firing is subject to the input conditions, denoted by token availability.
1
2
1a
1b
2
1
2a
2b
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
93


Extension with on input arcs
It is possible to defined a weight on input arcs: it is the number of token which are consumed
1
2
Arc Weight
(nota: default is 1)
2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
94

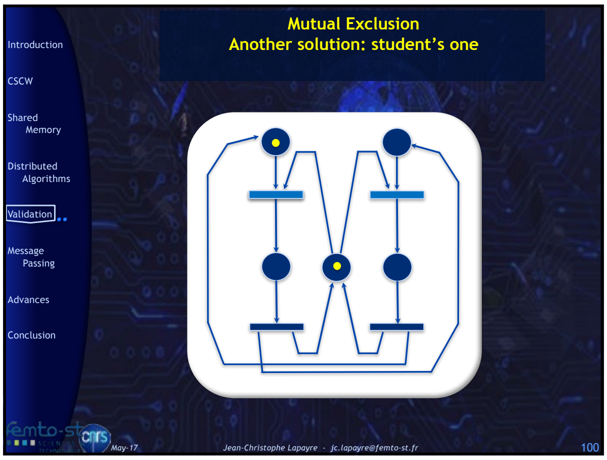

Several Distributed Algorithms Phases
Modelized by Petri Net.
Parallelism
Synchronization for Meeting (mpi_barrier)
Synchronization by Semaphore
Resource access
Resource access
Resources Sharing
Memorization
Capacity Limitation
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
95

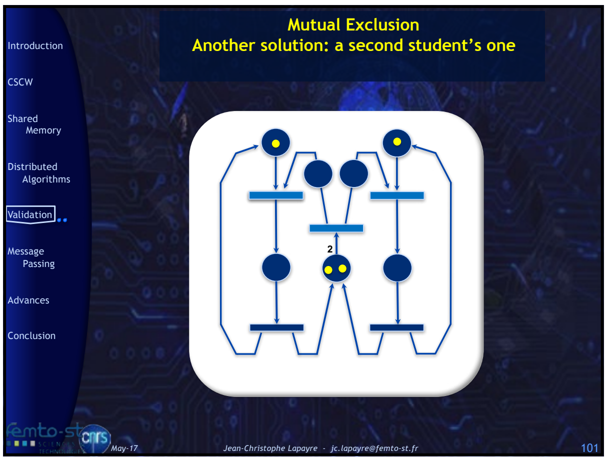

Concurrency
Independent inputs permit "concurrent" firing of transitions
1a
1b
2
1a
1b
2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
96

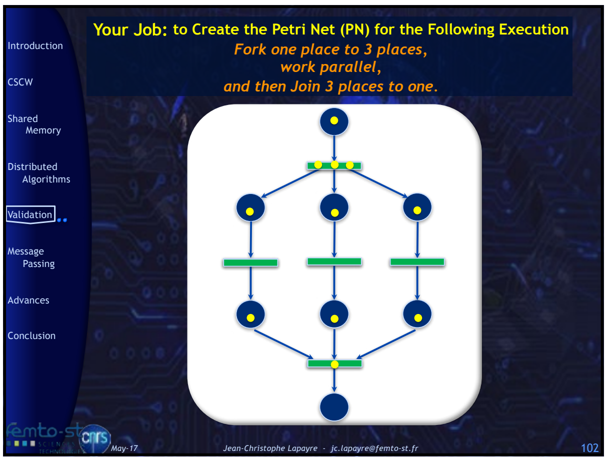

Conflict
Overlapping (sharing) inputs put transitions in conflict
1a
1b
1b
a
b
2
2
Only one of a or b may fire
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
97

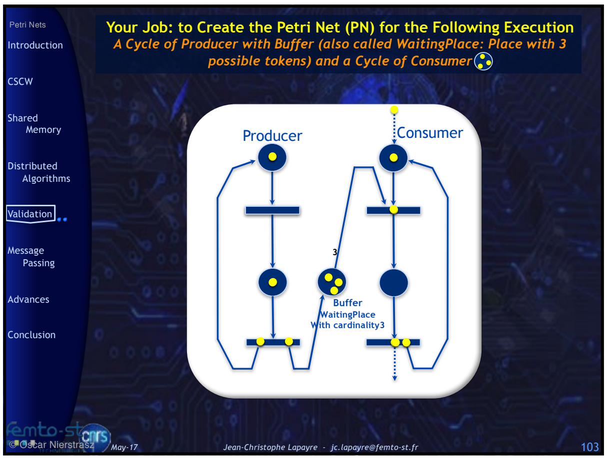

Mutual Exclusion
The two subnets are forced to synchronize
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
98

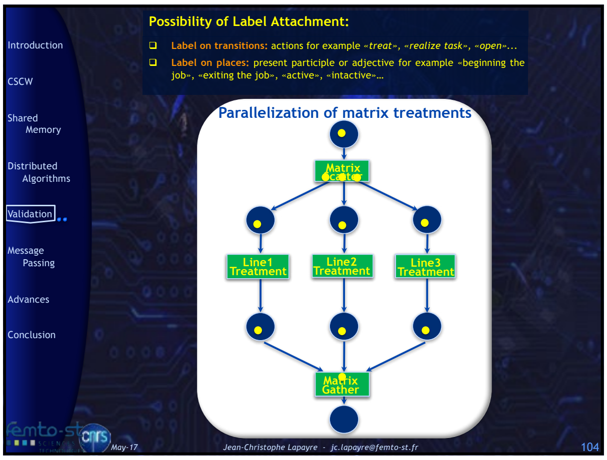

Mutual Exclusion
Your job: The two subnets are forced to synchronize with equity
2
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
99

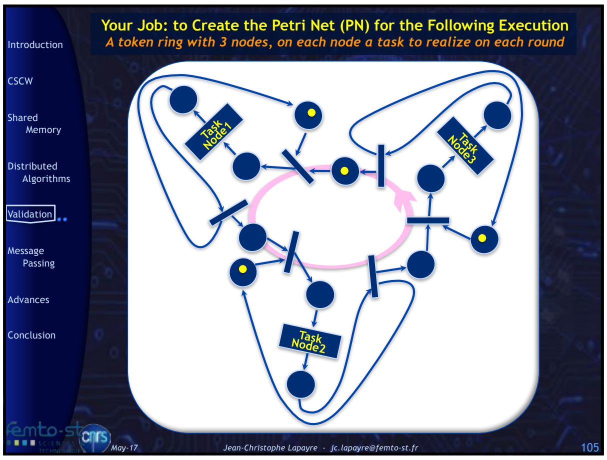

Mutual Exclusion
Another solution: student's one
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
100

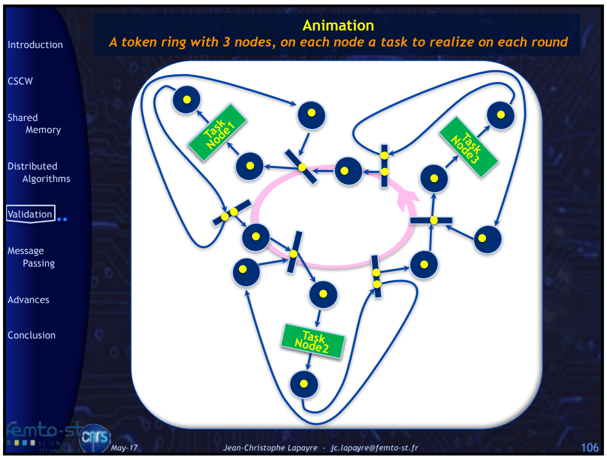

Mutual Exclusion
Another solution: a second student's one
2
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
101


Your Job: to Create the Petri Net (PN) for the Following Execution
Fork one place to 3 places,
work parallel,
and then Join 3 places to one.
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
102

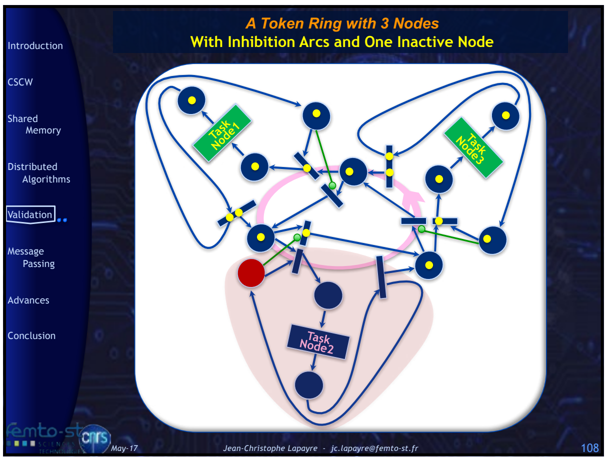

Petri Nets
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Your Job: to Create the Petri Net (PN) for the Following Execution
A Cycle of Producer with Buffer (also called WaitingPlace: Place with 3 possible tokens) and a Cycle of Consumer
Producer
Consumer
3
Buffer
WaitingPlace
With cardinality3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
103

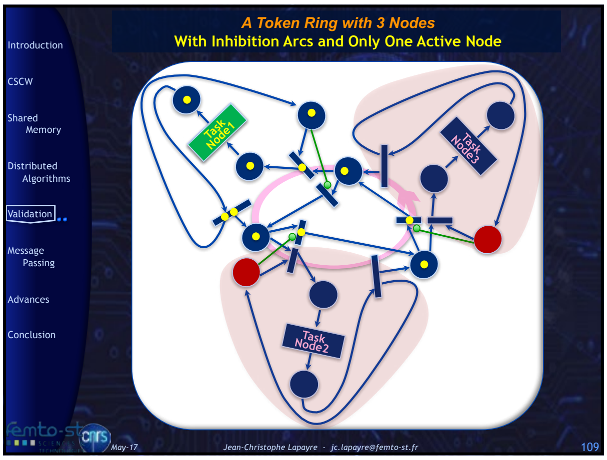

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Possibility of Label Attachment:
Label on transitions: actions for example «treat», «realize task», «open»…
Label on places: present participle or adjective for example «beginning the job», «exiting the job», «active», «intactive»…
Parallelization of matrix treatments
Matrix Scatter
Line1 Treatment
Line2 Treatment
Line3 Treatment
Matrix Gather
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
104

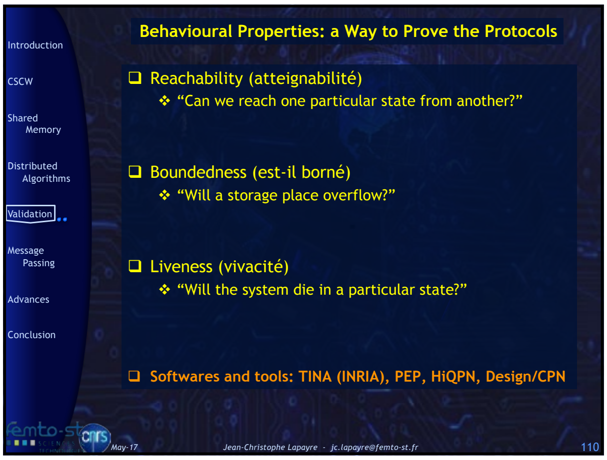

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Your Job: to Create the Petri Net (PN) for the Following Execution
A token ring with 3 nodes, on each node a task to realize on each round
Task Node1
Task Node3
Task Node2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
105


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Animation
A token ring with 3 nodes, on each node a task to realize on each round
Task Node1
Task Node3
Task Node2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
106

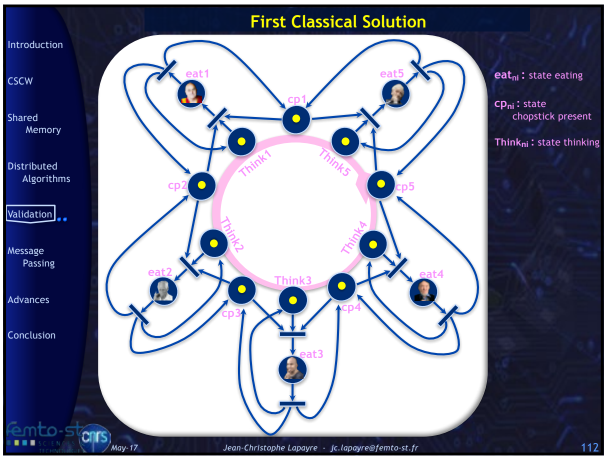

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Possible Declinaisons of Petri Net
1a
1b
2
Inhibitor arcs:
Inhibitor arcs are represented with a circle-headed arc.
The transition can fire iff the inhibitor place does not contain tokens.
Weight & Capacity: It is possible to define every place has a capacity and every arc has a weight.
Timed Petri Net:
Time delays associated with transitions and/or places.
Fixed delays or interval delays.
Stochastic Petri nets: exponentially distributed random variables as delays.
Coloured Petri Net: Tokens or places have "colours", holding complex information.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
107

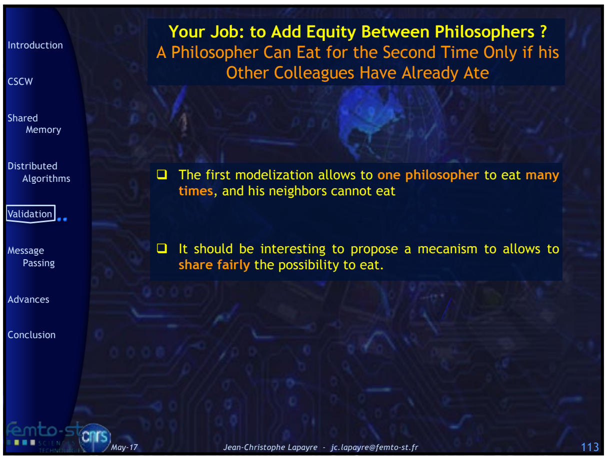

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
A Token Ring with 3 Nodes
With Inhibition Arcs and One Inactive Node
Task Node1
Task Node3
Task Node2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
108

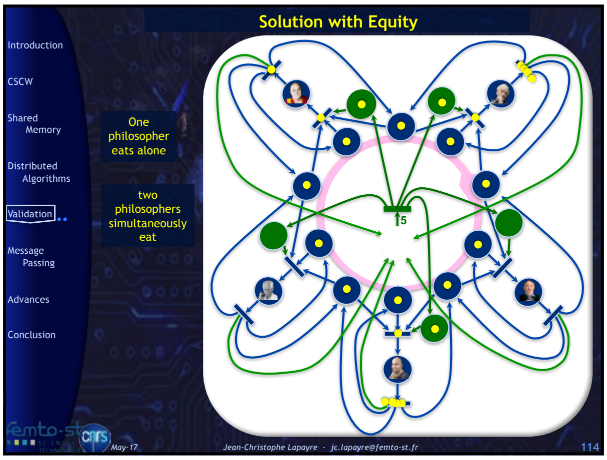

A Token Ring with 3 Nodes
With Inhibition Arcs and Only One Active Node
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Task Node1
Task Node2
Task Node3
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
109


Behavioural Properties: a Way to Prove the Protocols
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❑ Reachability (atteignabilité)
❖ "Can we reach one particular state from another?"
❑ Boundedness (est-il borné)
❖ "Will a storage place overflow?"
❑ Liveness (vivacité)
❖ "Will the system die in a particular state?"
❑ Softwares and tools: TINA (INRIA), PEP, HiQPN, Design/CPN
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
110


Your Job to conclude this Part on Petri Net:
The five Philosophers Welknown Problem
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
In 1965, Dijkstra posed and solved synchronization problem called the dining philosophers problem. problem can be stated as follows:
❑ Five philos rs are sittin table. Eac sopher h noddles.
❑ A philosoph eat it. between each plate
❑ The life of a philo alternate periods thinking.
❑ When a philosopher tries to acquire his chopsticks, one at time.
❑ If successful acquir chopsticks, a ph her eats while, then puts e chopsticks and continues to
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
111


First Classical Solution
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
eat1
eat2
eat3
eat4
eat5
cp1
cp2
cp3
cp4
cp5
Think1
Think2
Think3
Think4
Think5
eat$_{ni}$: state eating
cp$_{ni}$: state chopstick present
Think$_{ni}$: state thinking
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
112


Your Job: to Add Equity Between Philosophers ?
A Philosopher Can Eat for the Second Time Only if his Other Colleagues Have Already Ate
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❑ The first modelization allows to one philosopher to eat many times, and his neighbors cannot eat
❑ It should be interesting to propose a mecanism to allows to share fairly the possibility to eat.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
113


Solution with Equity
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
One philosopher eats alone
two philosophers simultaneously eat
5
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
114

Outline

Other Possible Modelization of Distributed Protocols

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

- Finite state automata are classicaly used to modelize distributed protocols.
- These automata are used to prove these protocols.
- A finite state automaton is composed of states and transitions.

First Example to Understand the Construction of an Automaton

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

- **Remote Teaching Scenario**
  - This is a collaborative example.
    - It is not a lecture course
    - This is a tutorial session: students can communicate with students and they can communicate with the teacher.

Behaviors Modelization

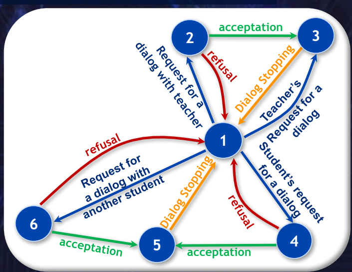
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

- To establish study a system, it is first necessary to define the behavior of the actors:
  - First Automata: the behavior of the students,
  - Second Automata: the behavior of the teacher.

Student Behavior

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

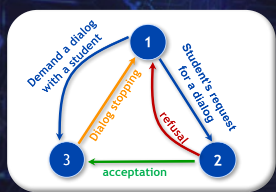
State1: Initial state
State2: Student requesting a dialog with the teacher
State3: Student in a dialog with teacher
State4: Student subject to a student's request for a dialog
State5: Student in a dialog with another student
State6: Student requesting a dialog with another student



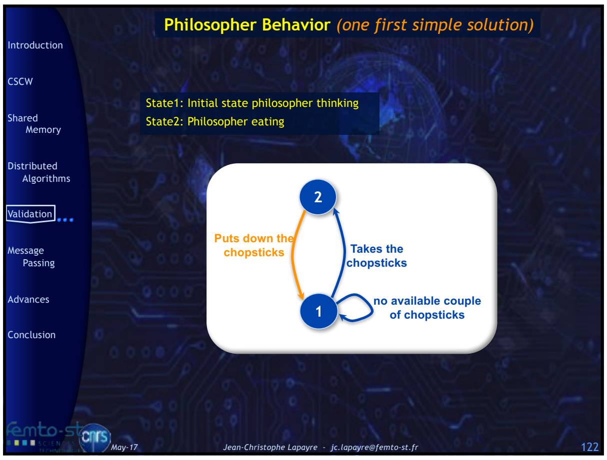
Teacher Behavior

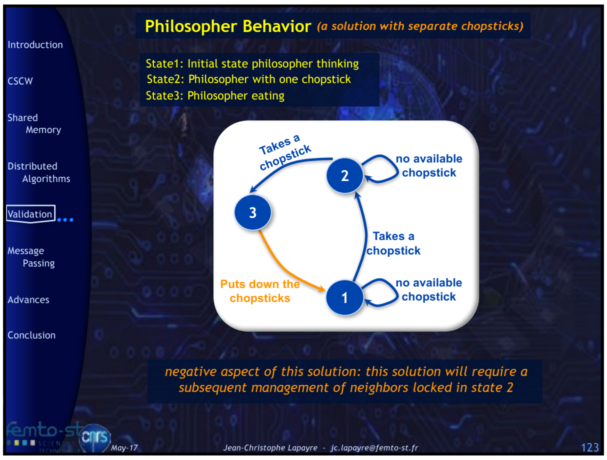
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

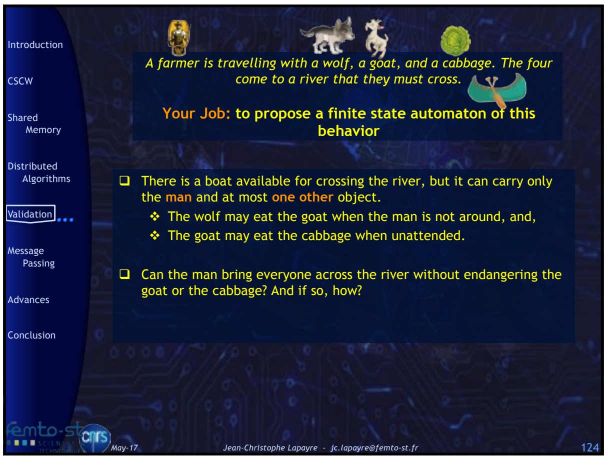
State1: Initial state
State2: Teacher subject to a student's request for a dialog
State3: Teacher in a dialog with a student

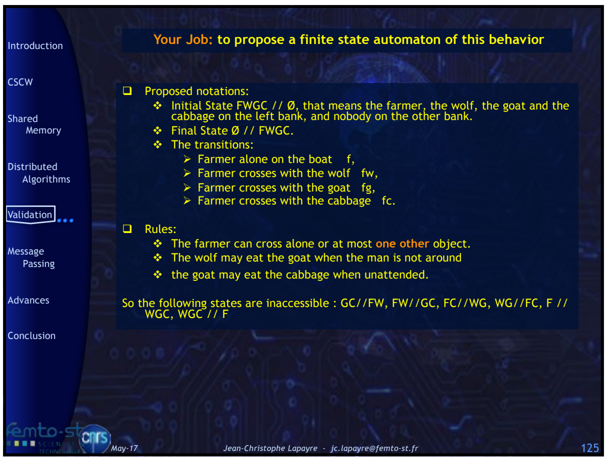


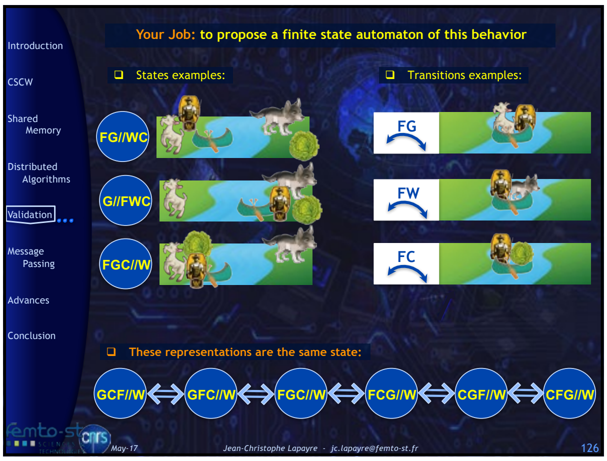

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
Do you remember *The five Philosophers ?*
Your job, this time: to design the automata of a philosopher behavior
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
121

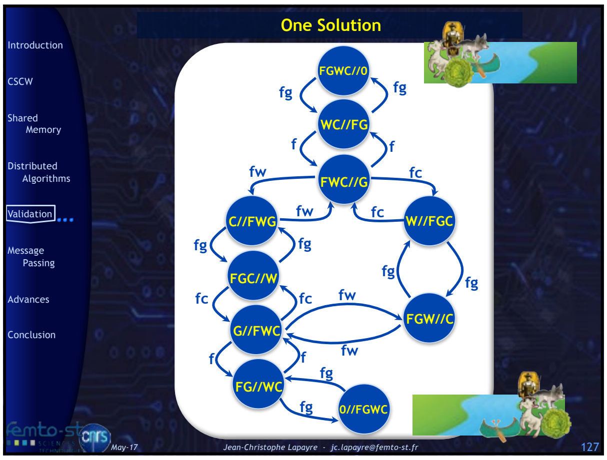

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
Philosopher Behavior *(one first simple solution)*
State1: Initial state philosopher thinking
State2: Philosopher eating
2
Puts down the chopsticks
Takes the chopsticks
1
no available couple of chopsticks
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
122

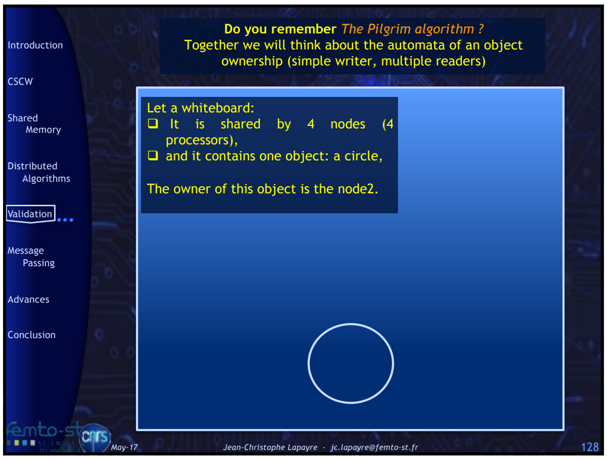

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
Philosopher Behavior *(a solution with separate chopsticks)*
State1: Initial state philosopher thinking
State2: Philosopher with one chopstick
State3: Philosopher eating
Takes a chopstick
2
no available chopstick
3
Takes a chopstick
Puts down the chopsticks
1
no available chopstick
negative aspect of this solution: this solution will require a subsequent management of neighbors locked in state 2
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
123


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
A farmer is travelling with a wolf, a goat, and a cabbage. The four come to a river that they must cross.
Your Job: to propose a finite state automaton of this behavior
❑ There is a boat available for crossing the river, but it can carry only the man and at most one other object.
❖ The wolf may eat the goat when the man is not around, and,
❖ The goat may eat the cabbage when unattended.
❑ Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
124

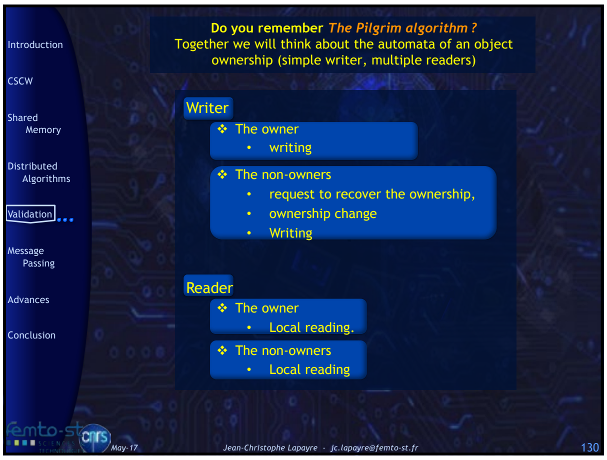

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
Your Job: to propose a finite state automaton of this behavior
❑ Proposed notations:
❖ Initial State FWGC // Ø, that means the farmer, the wolf, the goat and the cabbage on the left bank, and nobody on the other bank.
❖ Final State Ø // FWGC.
❖ The transitions:
➢ Farmer alone on the boat f,
➢ Farmer crosses with the wolf fw,
➢ Farmer crosses with the goat fg,
➢ Farmer crosses with the cabbage fc.
❑ Rules:
❖ The farmer can cross alone or at most one other object.
❖ The wolf may eat the goat when the man is not around
❖ the goat may eat the cabbage when unattended.
So the following states are inaccessible : GC//FW, FW//GC, FC//WG, WG//FC, F // WGC, WGC // F
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
125

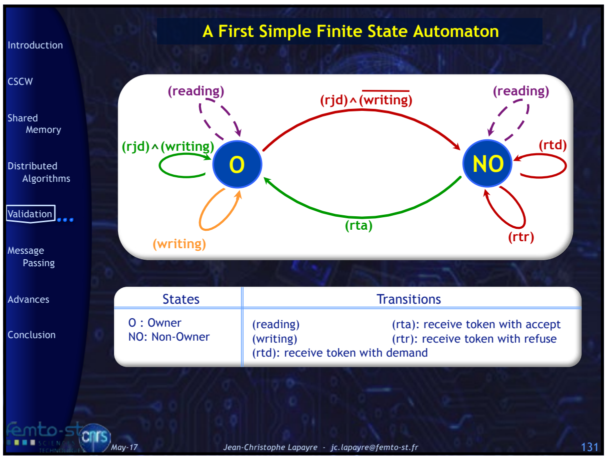

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion
Your Job: to propose a finite state automaton of this behavior
❑ States examples:
❑ Transitions examples:
FG//WC
G//FWC
FGC//W
FG
FW
FC
❑ These representations are the same state:
GCF//W
GFC//W
FGC//W
FCG//W
CGF//W
CFG//W
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
126

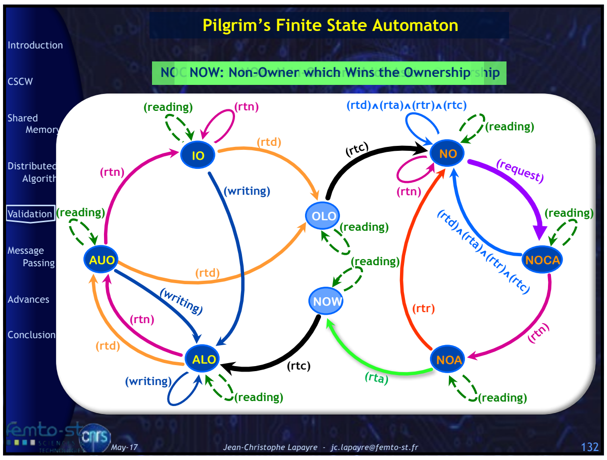

One Solution
FGWC//0
WC//FG
FWC//G
C//FWG
W//FGC
FGC//W
G//FWC
FGW//C
FG//WC
0//FGWC
fg
f
fw
fc
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
127


Do you remember The Pilgrim algorithm ?
Together we will think about the automata of an object ownership (simple writer, multiple readers)
Let a whiteboard:
It is shared by 4 nodes (4 processors),
and it contains one object: a circle,
The owner of this object is the node2.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
128


Do you remember The Pilgrim algorithm ?
Together we will think about the automata of an object ownership (simple writer, multiple readers)
3
Becomes the new owner of the circle
2
and accepts to lose the ownership
Node1
Node2
Node3
Node4
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
129


Do you remember The Pilgrim algorithm ?
Together we will think about the automata of an object ownership (simple writer, multiple readers)
Writer
The owner
writing
The non-owners
request to recover the ownership,
ownership change
Writing
Reader
The owner
Local reading.
The non-owners
Local reading
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
130


A First Simple Finite State Automaton
(reading)
(rjd)∧(writing)
O
NO
(rtd)
(rta)
(rtr)
(writing)
States
O : Owner
NO: Non-Owner
Transitions
(reading)
(writing)
(rtd): receive token with demand
(rta): receive token with accept
(rtr): receive token with refuse
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
131


Pilgrim's Finite State Automaton
NOW: Non-Owner which Wins the Ownership
IO
NO
OLO
AUO
NOCA
NOW
ALO
NOA
(reading)
(rtn)
(rtd)
(writing)
(rtc)
(rta)
(rtr)
(request)
(rtd)∧(rta)∧(rtr)∧(rtc)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
132

This Automaton Was Used to Prove these Theorems

❑ **Mutual Exclusion Theorem:**

The Pilgrim protocol guarantees mutual exclusion for writing on a shared object.

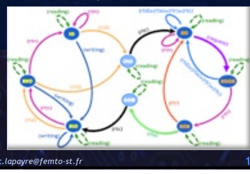
❑ **Liveliness Theorem:**

An object always has an owner or has one after finite time t.

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

Pilgrim's Proof

- ❑ Let E be the set of states containing n elements and T the set of transitions(events) containing m elements,
- ❑ The automaton edges represent an application from $E \times T \rightarrow E$
- ❑ The structure created is a graph with $n \cdot m$ edges.
- ❑ $n = 8$ et $m = 8$, so the mathematic definition imposes 64. But some of them are impossible.
- ❑ If we strictly abide by this definition, we have to create a trash state which would receive all *impossible edges*. To simplify our model we have not created this state.
- ❑ For each state we also state *10* rules defining the automaton, and especially the *impossible events*.



Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

Pilgrim's Proof: 10 rules

For each state we also state *10* rules defining the automaton, and especially the *impossible events*.

❑ Rule 1:
The (reading) event does not change the state of a site.

❑ Rule 2:
A site can write on an object only if it is in one of the following states: ALO (Active Locked Owner), AUO (Active Unlocked Owner) or IO (Inactive Owner).

❑ Rule 3:
A site can send an ownership request for an object only if it is in the NO (Non-Owner) state.

❑ Rule 4:
At time t one site at the most can be in the NOA (Non-Owner which Asks for the ownership) state.

❑ Rule 5:
A site in the NOA state cannot receive a message with an accept or refuse command (one of the two events (rta) ou (rtr))

❑ Rule 6:
A site in the NOW (Non-Owner which Wins the ownership) state or in the OLO (Owner which Loses the Ownership) state cannot receive a token with a request, an accept command or a refuse command (one of the following events (rtd), (rta) ou (rtr)).

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

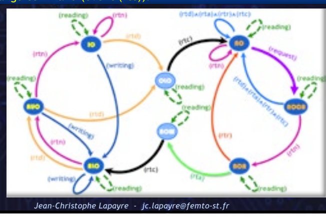
Pilgrim's Proof: 10 rules

❑ Rule 7:
A site in the NOW (Non-Owner which Wins the ownership) state or in the OLO (Owner which Loses the Ownership) state cannot receive a token with no command (event (rtn)).

❑ Rule 8:
A site in the NOW (Non-Owner which Wins the ownership) state or in the OLO (Owner which Loses the Ownership) state cannot write on the object.

❑ Rule 9 :
A site in one of the owner states (ALO, AUO, IO or OLO) cannot receive a token with an answer to a request (event (rta) or (rtr)).

❑ Rule 10:
A site in one of these three owner states (ALO, AUO, IO) cannot receive a token with an owner change command (event (rtc)).



Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

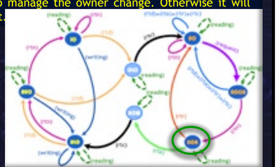
Pilgrim's Proof: examples of States Justifications

❑ **NOA (Non-Owner which Asks the ownership) State:**
Rule 5 recall:
A site in the NOA state cannot receive a message with an accept or refuse command (one of the two events (rta) ou (rtr))

Justification: In this state a site is waiting for an answer to its ownership earn request: accept command or refuse command. if a NOA site received a token with no command or a token with an ownership change command or an ownership earn request (one of the following events (rtn), (rtc) or (rtd)), this would indicate a dysfunctioning of the system like a fault of the owner that cannot send an answer.

Rules 2 and 3 show that in NOA the events (writing) and (request) are impossible. A (reading) event does not change the state of the site(rule 1).

If the ownership earn request is accepted, the site changes to NOW (Non-Owner which Wins the ownership), and will have to manage the owner change. Otherwise it will return to NO and make another request.



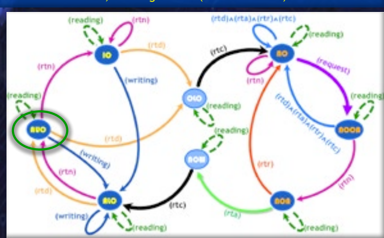
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation...
Message Passing
Advances
Conclusion

Pilgrim's Proof: examples of States Justifications

❑ **AUO (Active Unlocked Owner) State:**

Rule 3 shows that in AUO, the event (request) is impossible. A (reading) event does not change the state of the site (rule 1). Rules 9 and 10 show that the following events are impossible: (rta),(rtr) and (rtc). If a site in AUO writes on the object, it changes to ALO, because it becomes active again.

We called this state Active Unlocked Owner, because if it receives a token with a request (event (rtd)), it has to accept the request. Then, it changes to OLO when receiving a token with no command, it changes to IO (Inactive Owner).



Pilgrim: Proof of Mutual Exclusion Theorem

The Pilgrim protocol guarantees mutual exclusion for writing on a shared object.

Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation… · Message Passing · Advances · Conclusion

- Rule 11:
A non-ownership site cannot answer an ownership earn request.

- Lemme 1 :
When a site in the NOW (Non-Owner which Wins the ownership) state changes to the ALO (Active Locked Owner) state, the site which was the owner changes to the NO state.

- Lemme 2 :
When a site changes from the OLO (Owner which Looses the Ownership) state to the NO (Non-Owner) state, one and only one site becomes the owner of the object. As soon as the winner receives the token, it changes to ALO (Active Locked Owner).

Using Lemme1, and Lemme2, we prove:
- **Mutual Exclusion Theorem:**
The Pilgrim protocol guarantees mutual exclusion for writing on a shared object.

At time t_0, consider a site in ALO (Active Locked Owner) and $n\text{-}1$ sites in NO. The theorem is true at time t_0, because the ALO site is the only one which can write on the object. We have to prove that this theorem still remains true (for each event that modifies the system).

Conclusion: The mutual exclusion theorem is true at time t_0 and remains true following events which modify the system. Therefore the Pilgrim protocol guarantees mutual exclusion for writing on a shared object.

Pilgrim: Proof of Liveliness Theorem

An object always has an owner or has one after nite time t.

Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation… · Message Passing · Advances · Conclusion

Using Using the same approach, we have to show that the liveliness theorem is true at time t_0 and that it remains true following various events which modify the system.

At time t_0, consider a site in ALO (Active Locked Owner) and $n\text{-}1$ sites in NO. The theorem is initially true, because the ALO site is the owner of the object.

Working with Rule1, Rule 2, Lemme2 and Rule 4 the proof is written.

Conclusion: The liveliness theorem is true at time t_0 and remains true following events which modify the system. Therefore with the Pilgrim protocol an object always has an owner or has one after finite time t.

Outline

Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation · Message Passing · Advances · Conclusion

I. Computer Supported Collaborative Work

II. Shared Memory Theory

III. Distributed Algorithms for Shared Memory Management

IV. Validations of Distributed Algorithms

V. Implementations on Message Passing Layer
- PVM,
- MPI.

IV. Computer Science Advances in these Domains

PVM: Parallel Virtual Machine

Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation · Message Passing · Advances · Conclusion

- Version 1: 1989-90 Oak Ridge National Laboratory (first a research project),
- Version 2: 1991 ; more simple ; more robust,
- Version 3: 1993 ; is more suitable for parallel architectures,
- Version 3.4: Major extensions(Introduction of communicative contextsand static process groups).

- PVM provides an unified framework for developing parallel programs with the existing infrastructure,
- PVM enables a collection of heterogeneous computer systems as a single parallel virtual machine,
- Transparent to the user,

- All tasks on PVM cooperate by sending and receiving messages from one another,
- PVM supports functional and data parallelism,
- A well defined library of PVM interface routines are used for programming.

PVM allows to consider a set of machines as an only one virtual machine with many processors *(virtual or not)*

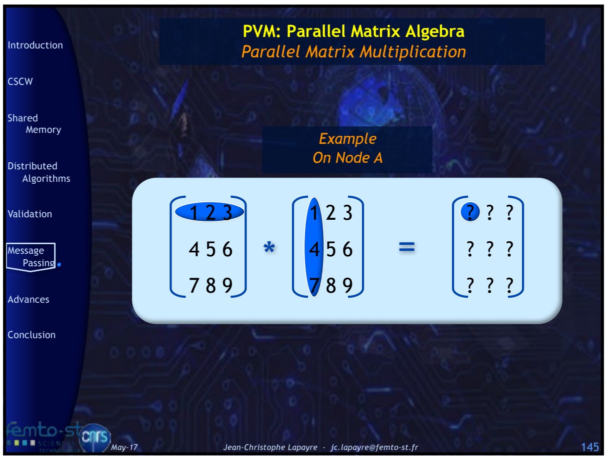
The first solution for GRID COMPUTING

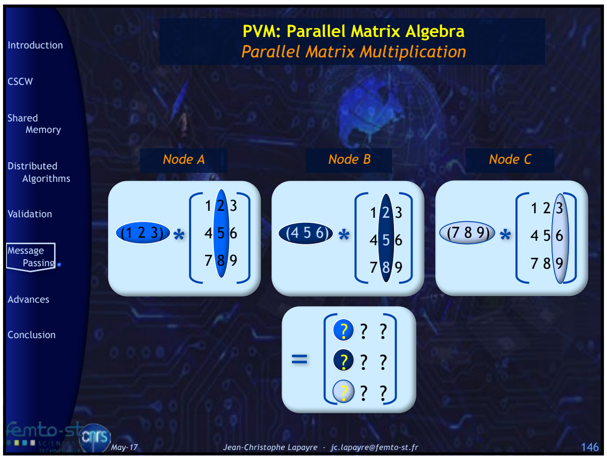
Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation · Message Passing · Advances · Conclusion

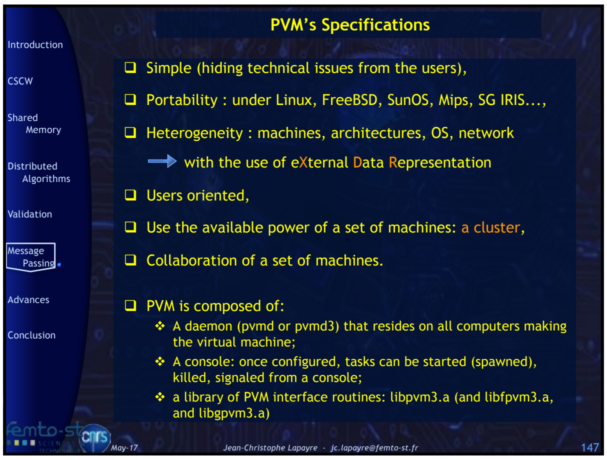
Many processors *virtual or not*

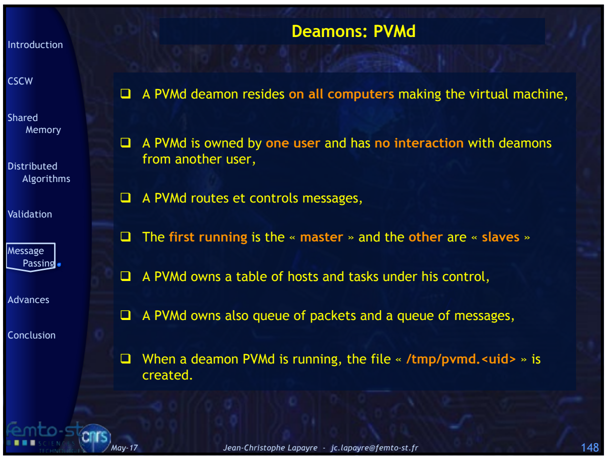
Introduction · CSCW · Shared Memory · Distributed Algorithms · Validation · Message Passing · Advances · Conclusion

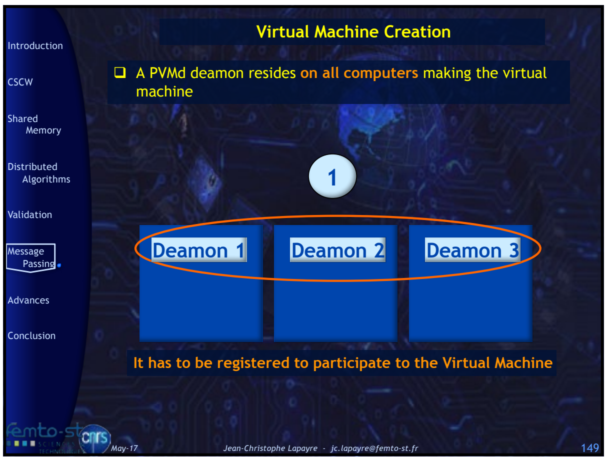
- PVM is used for the development of parallel programs or parallel algorithms:
  - Several virtual processors can be simulated on only one physical processor,
  - Or for real parallelism: each virtual processor is associated to a physical processor,
  - Mixed solutions with more or less physical and logical processors processors

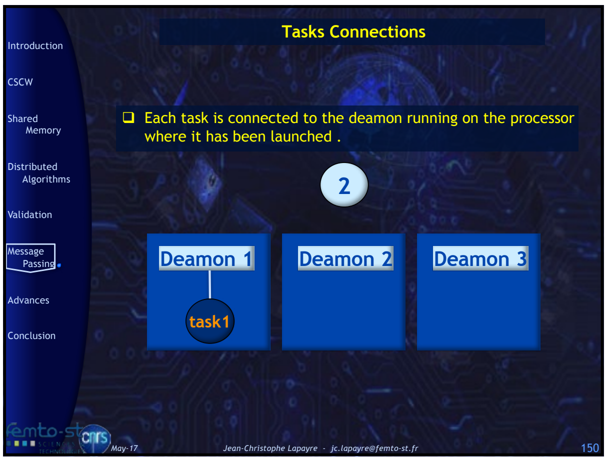

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
PVM: Parallel Matrix Algebra
Parallel Matrix Multiplication
Example
On Node A
1 2 3
4 5 6
7 8 9
*
1 2 3
4 5 6
7 8 9
=
? ? ?
? ? ?
? ? ?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
145


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
PVM: Parallel Matrix Algebra
Parallel Matrix Multiplication
Node A
Node B
Node C
(1 2 3) *
1 2 3
4 5 6
7 8 9
(4 5 6) *
1 2 3
4 5 6
7 8 9
(7 8 9) *
1 2 3
4 5 6
7 8 9
=
? ? ?
? ? ?
? ? ?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
146


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
PVM's Specifications
Simple (hiding technical issues from the users),
Portability : under Linux, FreeBSD, SunOS, Mips, SG IRIS…,
Heterogeneity : machines, architectures, OS, network
with the use of eXternal Data Representation
Users oriented,
Use the available power of a set of machines: a cluster,
Collaboration of a set of machines.
PVM is composed of:
A daemon (pvmd or pvmd3) that resides on all computers making the virtual machine;
A console: once configured, tasks can be started (spawned), killed, signaled from a console;
a library of PVM interface routines: libpvm3.a (and libfpvm3.a, and libgpvm3.a)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
147


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Deamons: PVMd
A PVMd deamon resides on all computers making the virtual machine,
A PVMd is owned by one user and has no interaction with deamons from another user,
A PVMd routes et controls messages,
The first running is the « master » and the other are « slaves »
A PVMd owns a table of hosts and tasks under his control,
A PVMd owns also queue of packets and a queue of messages,
When a deamon PVMd is running, the file « /tmp/pvmd.<uid> » is created.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
148

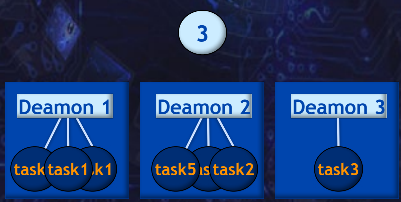

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Virtual Machine Creation
A PVMd deamon resides on all computers making the virtual machine
1
Deamon 1
Deamon 2
Deamon 3
It has to be registered to participate to the Virtual Machine
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
149


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Tasks Connections
Each task is connected to the deamon running on the processor where it has been launched .
2
Deamon 1
Deamon 2
Deamon 3
task1
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
150

Tasks Connections

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- A connected task can create other task (with pvm_spawn)



Hostfile, and console

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- The configuration of the virtual Machine is defined in the hostfile:
  - pvmd3 hostfile &
  - > pvm *to run the consol*
  - pvm> conf give the configuration of the Virtual Machine
- Console commands

```
pvm> help
help   - Print helpful information about a command
Syntax:  help [ command ]
Commands are:
  add    - Add hosts to virtual machine
  delete - Delete hosts from virtual machine
  halt   - Stop pvmds
  jobs   - Display list of running jobs
  kill   - Terminate tasks
  mstat  - Show status of hosts
  ps     - List tasks
  pstat  - Show status of tasks
  quit   - Exit console
  reset  - Kill all tasks
  spawn  - Spawn task
  version - Show libpvm version
  ...
pvm>
```


PVM program - Example

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

It Contains both parent and child code.

/ find out my task id number */*
mytid = pvm_mytid();

This function must be called before any other call. This call must be positive number.

pvm_perror(argv[0]);

This call tells what goes wrong with the last call.

/ find my parent's task id number */*
myparent = pvm_parent();

Retuns the Parent ID or else will retrn Error code "PvmNoParent" says that the process is spawned by user and not by any other process.

PVM program - Example

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- Following code differenciate the parent of all childs.
  / if i don't have a parent then i am the parent */*
  if (myparent == PvmNoParent) {
- So by this you can differenciate the code of top process and child code.
  / spawn the child tasks */*
  *info = pvm_spawn(argv[0], (char**)0, PvmTaskDefault, (char*)0, ntask, child);*
- This is the code which spawns childs.

PVM program - Example

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

/ spawn the child tasks */*
*info = pvm_spawn(argv[0], (char**)0, PvmTaskDefault, (char*)0, ntask, child);*

Parameters
- First perameter gives the name of the program that the child should run.
- Second perameter is the arguments that can be passed to child
- Third perameter - this says that where the child should spawn. We can give specific architecture or host name where the child can spawn
- Fourth perameter will have the values of the third perameter
- Fifth perameter specifies the number of the childs to be spawned.
- Last perameter is the array name which holds the child Id's.

This Function is returning the no of child successfully spawned which should be equal to the sent number n the 5th perameter other wise something gone wrong.

PVM program - Example

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

Following code is the waiting blocking call of parent.
/ recv a message from any child process */*
buf = pvm_recv(-1, JOINTAG);

So by this parent is waiting for all the childs to send the join message.
Parent gets the data from the child via the message and the following code gets the data from the buffer.
info = pvm_bufinfo(buf, &len, &tag, &tid);

After getting the join messages from all the childs parent also exits pvm.

PVM program - Example

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- Following code is run by child.

  info = pvm_initsend(PvmDataDefault);
  info = pvm_pkint(&mytid, 1, 1);
  info = pvm_send(myparent, JOINTAG);
  pvm_exit();

- First line - For message to be sent first we need to create the buffer to send the data.
- Second line - We are adding the child's Id to the message.
- Third line sending the join call to the parent with the message attached.
- Fourth line calls off the child from pvm and distroy its memory references.

Distributions of PVM

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- PVM debuggers
  - TotalView - commercial parallel debugger from Etnus (formerly Dolphinics) - well done!
  - Xmdb - parallel programming and debugging trainer for beginners
  - p2d2 - a portable parallel distributed debugger from NASA.
  - AIMS - nice tool developed by NASA
- Noteable PVM related Software
  - LPVM LPVM is PVM3 bindings for Common Lisp,
  - SCALAPACK - a library of optimized, parallel linear algebra routines using PVM,
  - pypvm-0.92 Python-PVM into a single release.,
  - IDL to PVM interface. lets you perform parallel processing with IDL through PVM call,
  - EasyPVM is a C++ wrapper for the PVM libraries,
  - PVM Toolbox for Matlab a toolkit for calling PVM from Matlab,
  - HP-PVM – PVM clone: Supports PVM 3.3 on Windows and Unix as well as shared memory,
  - Fortran 90 PVM interface - to take advantage of Fortran 90 facilities,
  - tkpvm - this package combines the power of tcl/tk and PVM,
  - WPVM 2.0 - a PVM version for Microsoft Windows,
  - jPVM - with PVM 3.4, a native methods interface to PVM for the Java (tm) platform,
  - JPVM - is a PVM-like class library implemented in and for use with Java,
  - Perl-PVM - Perl extension for PVM,
  - Pypvm is a Python interface to PVM,
  - CPPvm (C Plus Plus PVM) - a C++ interface to PVM 3.4,
  - …

Communications

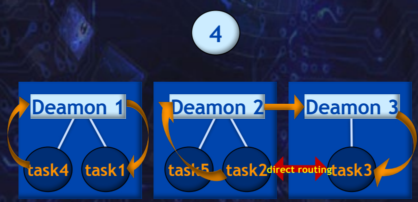
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- Communications are based on TCP and UDP/IP: UDP mode between deamons, possibility of TCP mode between tasks.
- Several communication modes are allowed:
  - Pvmd-pvmd
  - pvmd-task
  - task-task

Messages Exchanges

Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

- All connected tasks can communicate



Communications

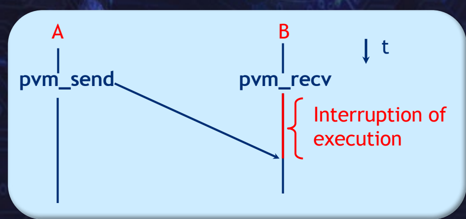
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

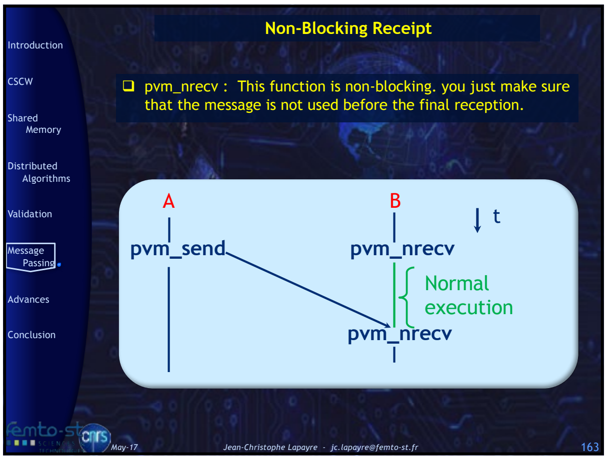
- Communications are asynchronous and keep the order of messages:
  - send: non-blocking
  - blocking receive
  - non-blocking receive
  - direct routing
  - multicasting and broadcasting

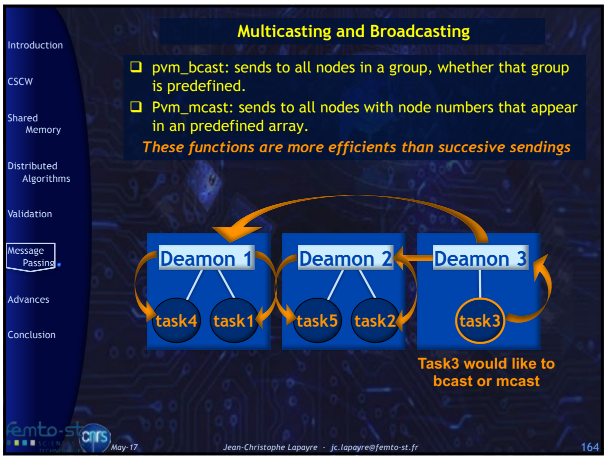
Blocking Receipt

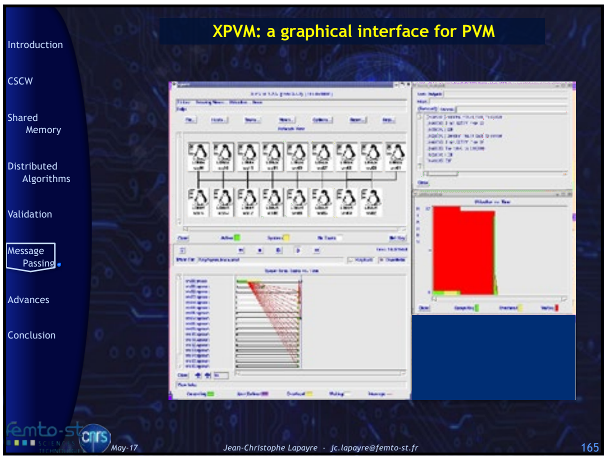
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion

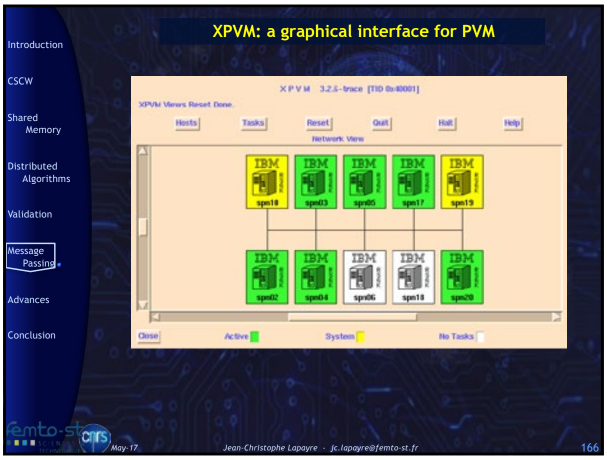
- pvm_recv: when this function is called the task is blocked until the reception of the message.

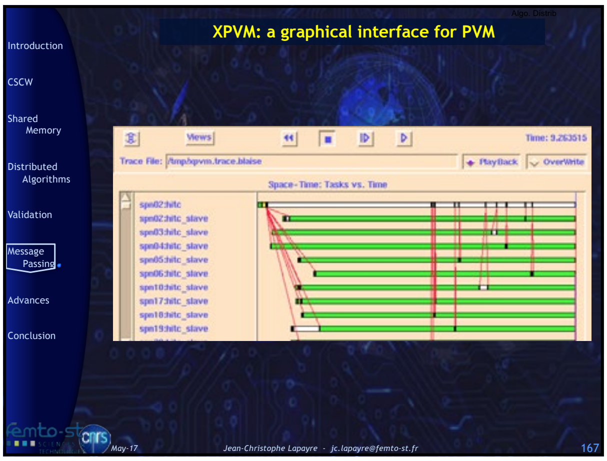


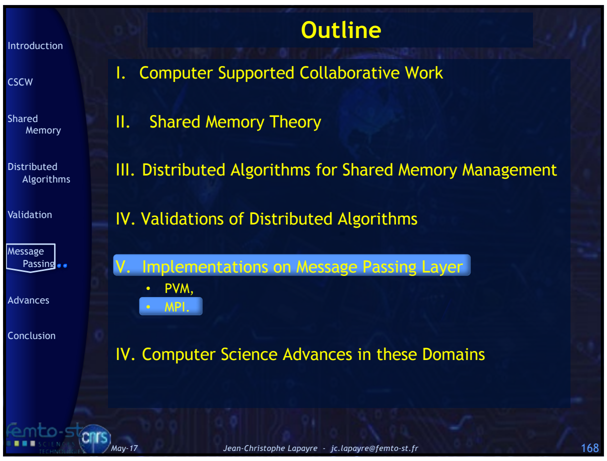

Non-Blocking Receipt
pvm_nrecv : This function is non-blocking. you just make sure that the message is not used before the final reception.
A
B
t
pvm_send
pvm_nrecv
Normal execution
pvm_nrecv
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
163


Multicasting and Broadcasting
pvm_bcast: sends to all nodes in a group, whether that group is predefined.
Pvm_mcast: sends to all nodes with node numbers that appear in an predefined array.
These functions are more efficients than succesive sendings
Deamon 1
Deamon 2
Deamon 3
task4
task1
task5
task2
task3
Task3 would like to bcast or mcast
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
164


XPVM: a graphical interface for PVM
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
165


XPVM: a graphical interface for PVM
Hosts
Tasks
Reset
Quit
Halt
Help
Network View
Close
Active
System
No Tasks
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
166


XPVM: a graphical interface for PVM
Views
Time: 9.263515
Trace File: /tmp/xpvm.trace.blaise
PlayBack
OverWrite
Space-Time: Tasks vs. Time
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
167


Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
PVM,
MPI.
IV. Computer Science Advances in these Domains
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
168

MPI

- You will work on MPI during practical exercises with Violeta Felea

PVM Versus MPI

PVM

- The development of PVM started in summer 1989 at Oak Ridge National Laboratory (ORNL).
- PVM was effort of a single research group, allowing it great flexibility in design of this system

MPI

- The development of MPI started in April 1992.
- MPI was designed by the MPI Forum (a diverse collection of implementors, library writers, and end users) quite independently of any specific implementation.

MPI-1 MPI-2

PVM-1 PVM-2 PVM-3 PVM-3.4

1989 90 94 96 97 99 2000

PVM Versus MPI

PVM	MPI
A distributed operating system	A library for writing application program, not a distributed operating system
Portability	Portability
	High Performance
Heterogeneity	Heterogeneity
Handling communication failures	Well-defined behavior

What is MPI ?

MPI - Message Passing Interface

- A fixed set of processes is created at program initialization, one process is created per processor :
  mpirun -np 5 program
- *Each process knows its personal number (rank)*
- *Each process knows number of all processes*
- *Each process can communicate with other processes*
- *Process cannot create new processes (in MPI-1)*

What is PVM ?

PVM - Parallel Virtual Machine

- Is a software package that allows an heterogeneous collection of workstations (host pool) to run as a single high performance parallel machine (virtual),
- PVM, through its virtual machine provides a simple yet useful distributed operating system,
- It has daemons running on all computers making up the virtual machine.

What is Not Different?

- *Portability* - source code written for one architecture can be copied to a second architecture, compiled and executed without modification (to some extent),
- Support *MPMD* (Multiple Program Mutiple Data) programs as well as *SPMD* (Simple Program Multiple Data),
- *Interoperability* - the ability of different implementations of the same specification to exchange messages,
- *Heterogeneity* (to some extent)
  PVM & MPI are systems designed to provide users with libraries for writing portable, heterogeneous, MPMD programs.

Process control: PVM and MPI

- ❑ Ability to start and stop tasks, to find out which tasks are running, and possibly where they are running,
- ❑ *PVM* contains all of these capabilities – it can spawn/kill tasks dynamically,
- ❑ *MPI -1* has no defined method to start new task.
- ❑ *But MPI -2* contain functions to start a group of tasks and to send a kill signal to a group of tasks,
- ❑ Message Passing operations: MPI Vs PVM:
  - ❖ *PVM*: Simple message passing,
  - ❖ *MPI* : Rich message support.

Conclusion: *PVM* Versus *MPI*

Each API Has its Unique Strengths

PVM

- ❑ Virtual machine concept
- ❑ Simple message passing
- ❑ Communication topology unspecified
- ❑ Interoperate across host architecture boundaries
- ❑ Portability over performance
- ❑ Resource and process control
- ❑ Robust fault tolerance

PVM is better for:
Heterogeneous cluster, resource and process control
The size of cluster and the time of program's execution are great

MPI

- ❑ No such abstraction
- ❑ Rich message support
- ❑ Support logical communication topologies
- ❑ Some realizations do not interoperate across architectural boundaries
- ❑ Performance over flexibility
- ❑ Primarily concerned with messaging
- ❑ More susceptible to faults

MPI is better for:
Supercomputers (PVM is not supported)
Max performance
Application needs rich message support

Outline

I. Computer Supported Collaborative Work

II. Shared Memory Theory

III. Distributed Algorithms for Shared Memory Management

IV. Validations of Distributed Algorithms

V. Implementations on Message Passing Layer

IV. Computer Science Advances in these Domains
- Consensus
- Adaptability,
- Active Networks,
- Components Platforms,
- Ontologies and Traceability

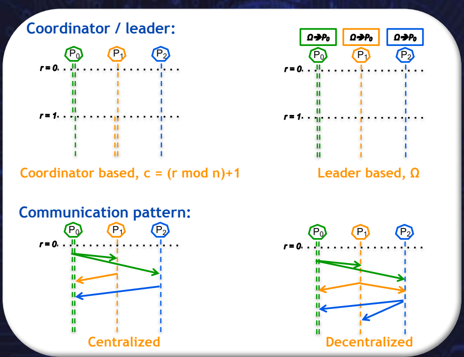
Consensus: Mutiple Writers Model

- ❑ Agree on a common value chosen from a group of proposed values:
  - ❖ Termination.
  - ❖ Agreement.
  - ❖ Validity.
- ❑ FLP theorem: [FISCHER, LYNCH and PATERSON 1985]:

Asynchronous system + process failure
→ impossible to solve the consensus problem

- ❑ Unreliable failure detectors : CHANDRA-TOEUG 1996.
  - •S → list of suspected processes.
- ❑ Leader oracle Ω → one correct process.
- ❑ Work in Crash-Stop model.

Existing consensus algorithms

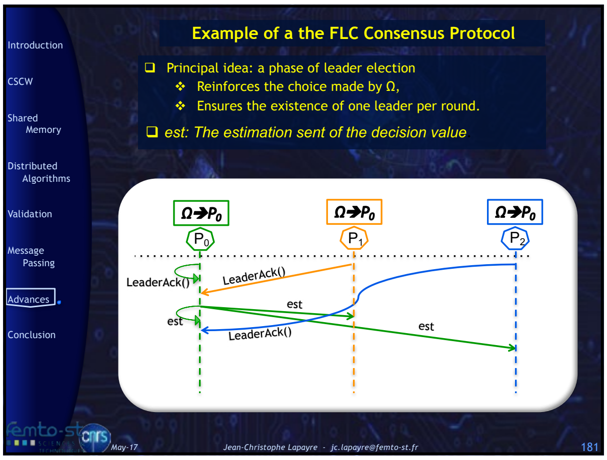


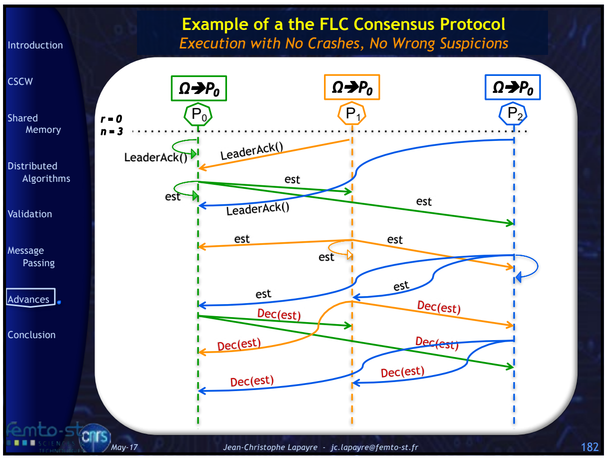
Existing consensus algorithms

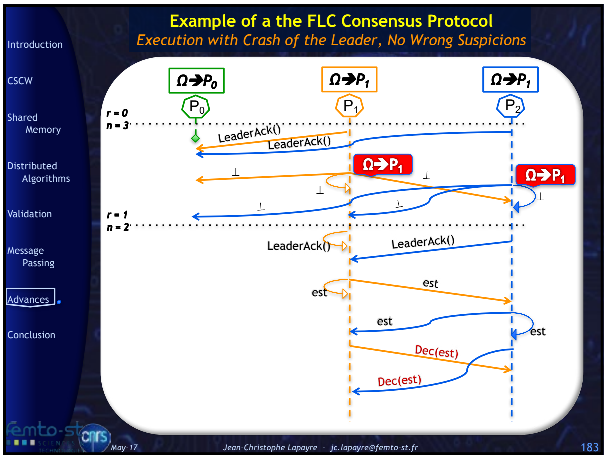
Classification

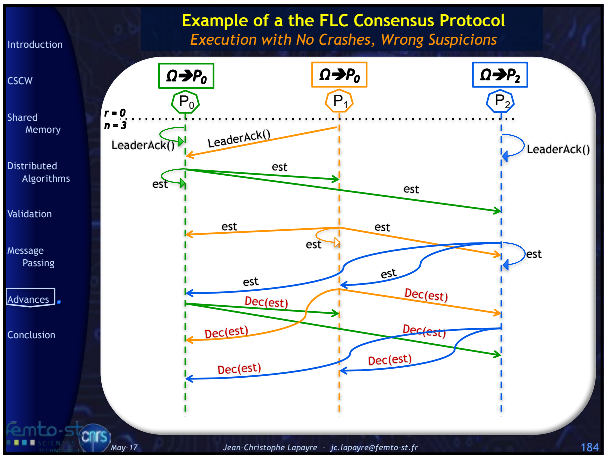
		Coordinator / leader	
		Coordinator	*Leader*
Communication pattern	*Centralized*	CT (*T. Chandra, S. Toeug*)	Paxos, MPaxos (*L. Lamport*)
	Decentralized	MR (*A. Mostefaoui, M. Raynal*)	MRLeader (*A. Mostefaoui, M. Raynal*)

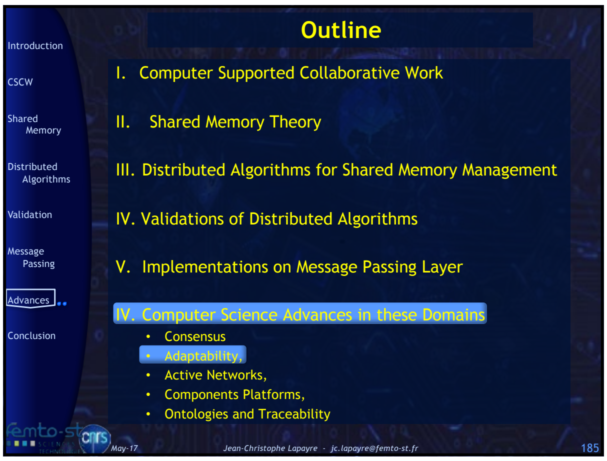
- ❑ Number of correct processes >= ⌈(n+1)/2⌉
  → tolerate f < n/2

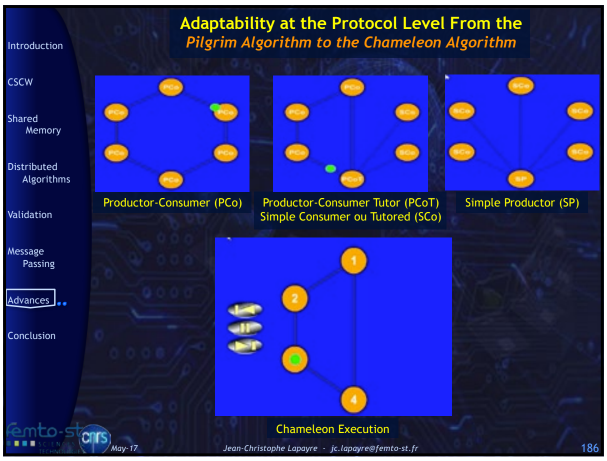

Example of a the FLC Consensus Protocol
Principal idea: a phase of leader election
Reinforces the choice made by Ω,
Ensures the existence of one leader per round.
est: The estimation sent of the decision value
Ω→P0
P0
P1
P2
LeaderAck()
est
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
181


Example of a the FLC Consensus Protocol
Execution with No Crashes, No Wrong Suspicions
Ω→P0
P0
P1
P2
r = 0
n = 3
LeaderAck()
est
Dec(est)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
182

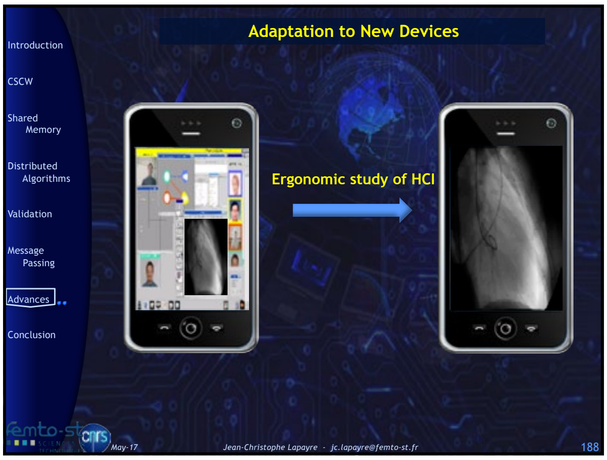

Example of a the FLC Consensus Protocol
Execution with Crash of the Leader, No Wrong Suspicions
Ω→P0
Ω→P1
P0
P1
P2
r = 0
n = 3
r = 1
n = 2
LeaderAck()
est
Dec(est)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
183


Example of a the FLC Consensus Protocol
Execution with No Crashes, Wrong Suspicions
Ω→P0
Ω→P2
P0
P1
P2
r = 0
n = 3
LeaderAck()
est
Dec(est)
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
184

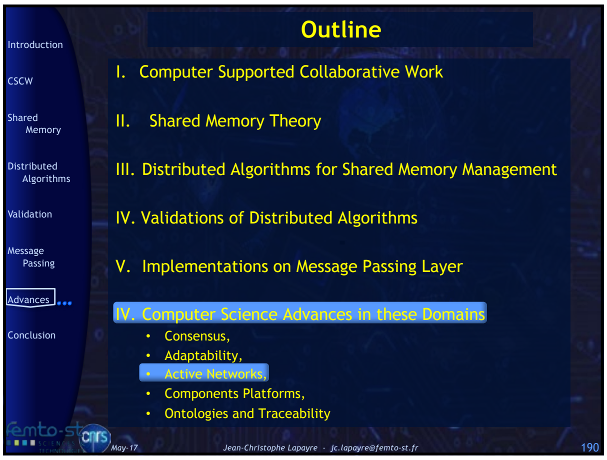

Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
IV. Computer Science Advances in these Domains
Consensus
Adaptability,
Active Networks,
Components Platforms,
Ontologies and Traceability
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
185

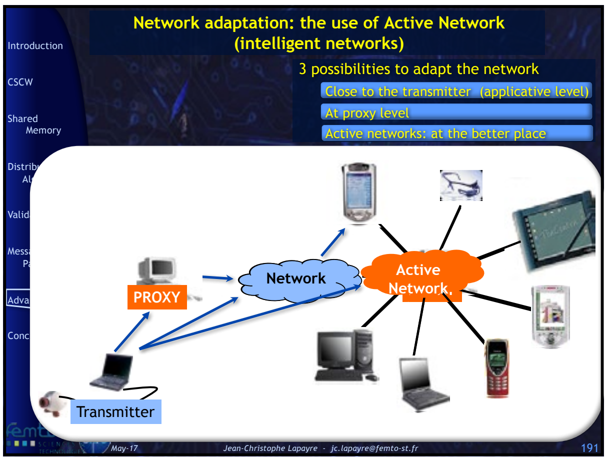

Adaptability at the Protocol Level From the
Pilgrim Algorithm to the Chameleon Algorithm
Productor-Consumer (PCo)
Productor-Consumer Tutor (PCoT)
Simple Consumer ou Tutored (SCo)
Simple Productor (SP)
Chameleon Execution
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
186

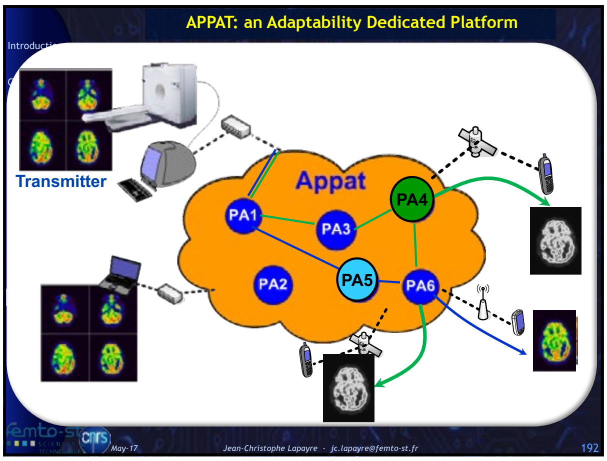

HCI Adaptability
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
187

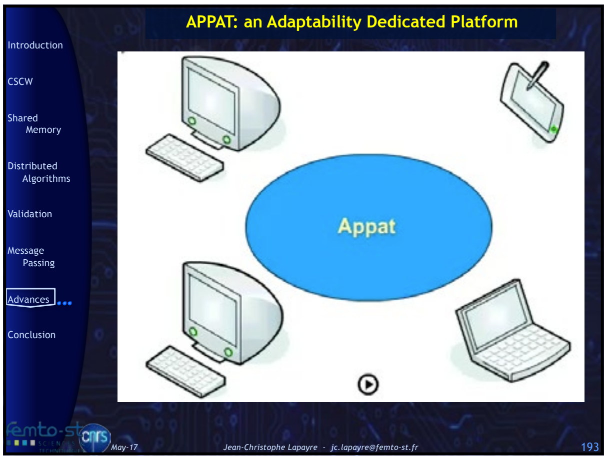

Adaptation to New Devices
Ergonomic study of HCI
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
188

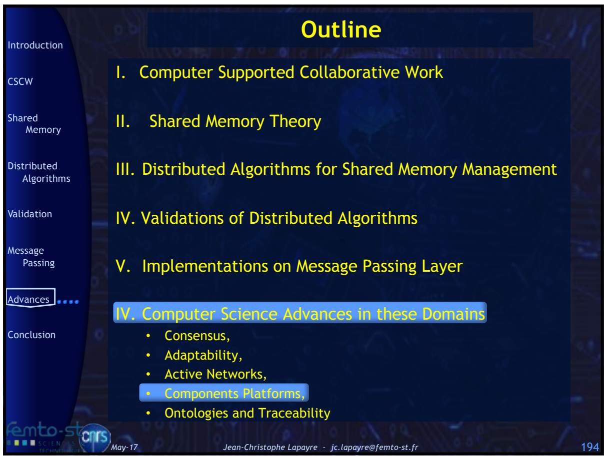

Ergonomic Adaptation of HCI, … plasticity ?
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
189

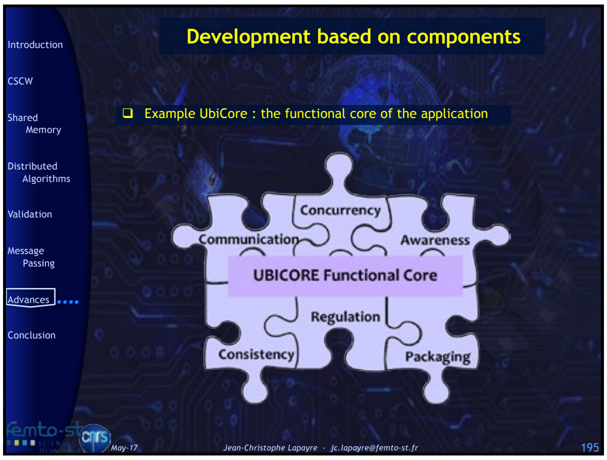

Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
IV. Computer Science Advances in these Domains
Consensus,
Adaptability,
Active Networks,
Components Platforms,
Ontologies and Traceability
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
190

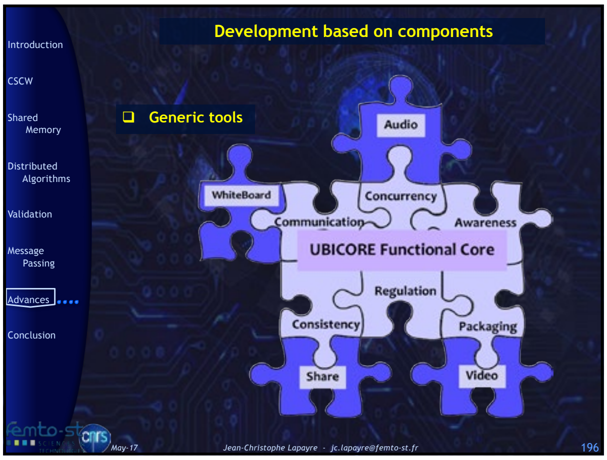

Network adaptation: the use of Active Network (intelligent networks)
3 possibilities to adapt the network
Close to the transmitter (applicative level)
At proxy level
Active networks: at the better place
PROXY
Network
Active Network
Transmitter
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
191

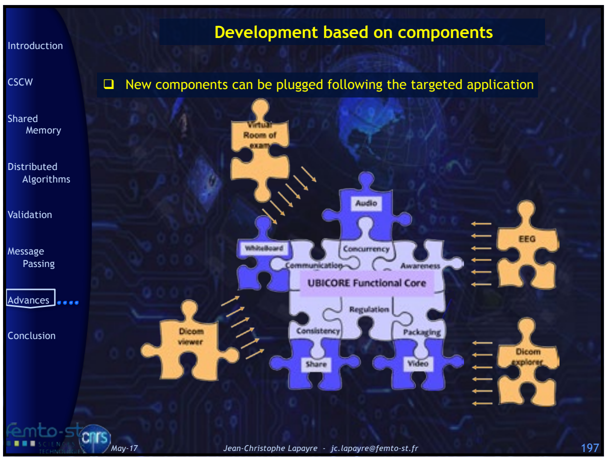

APPAT: an Adaptability Dedicated Platform
Transmitter
Appat
PA1
PA2
PA3
PA4
PA5
PA6
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
192

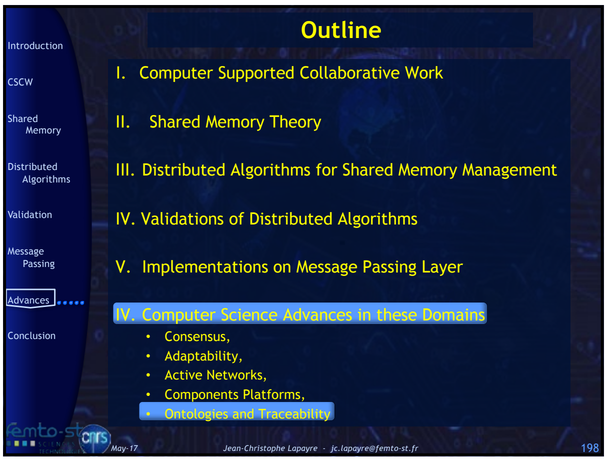

APPAT: an Adaptability Dedicated Platform
Appat
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
193

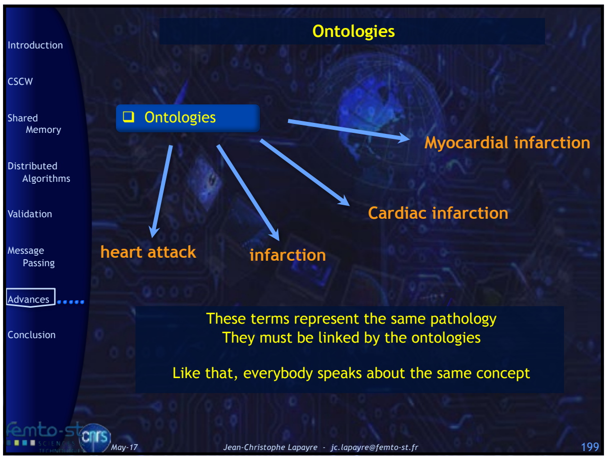

Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
IV. Computer Science Advances in these Domains
Consensus,
Adaptability,
Active Networks,
Components Platforms,
Ontologies and Traceability
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
194

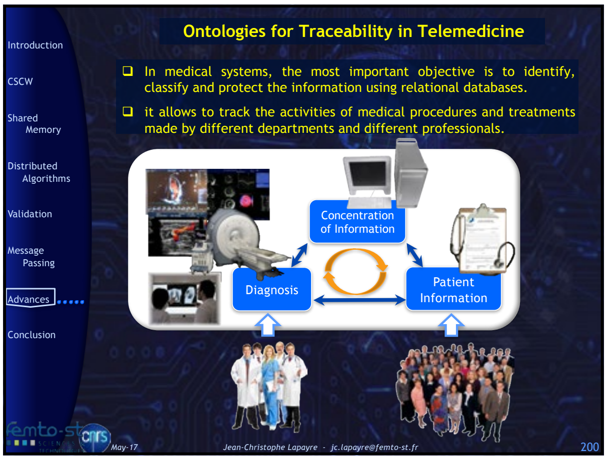

Development based on components
Example UbiCore : the functional core of the application
Concurrency
Communication
Awareness
UBICORE Functional Core
Regulation
Consistency
Packaging
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
195

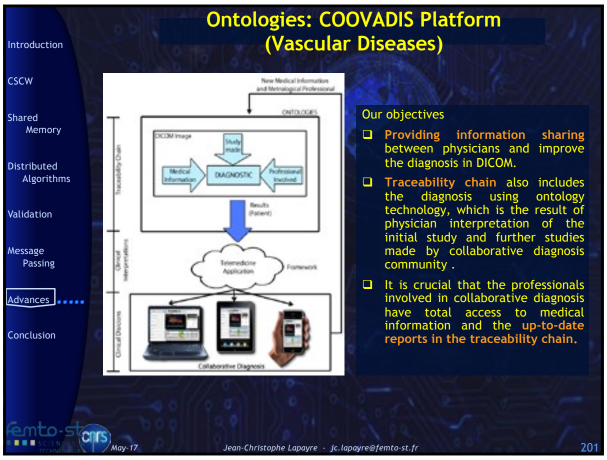

Development based on components
Generic tools
Audio
WhiteBoard
Concurrency
Communication
Awareness
UBICORE Functional Core
Regulation
Consistency
Packaging
Share
Video
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
196

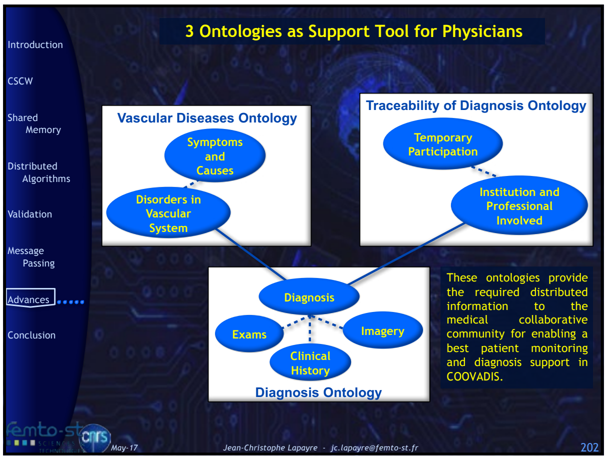

Development based on components
New components can be plugged following the targeted application
Audio
WhiteBoard
Concurrency
Communication
Awareness
UBICORE Functional Core
Regulation
Consistency
Packaging
Share
Video
EEG
Dicom viewer
Dicom explorer
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
197

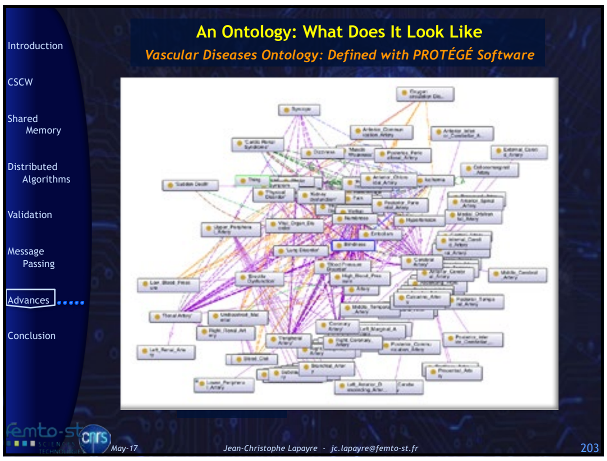

Outline
I. Computer Supported Collaborative Work
II. Shared Memory Theory
III. Distributed Algorithms for Shared Memory Management
IV. Validations of Distributed Algorithms
V. Implementations on Message Passing Layer
IV. Computer Science Advances in these Domains
Consensus,
Adaptability,
Active Networks,
Components Platforms,
Ontologies and Traceability
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
198

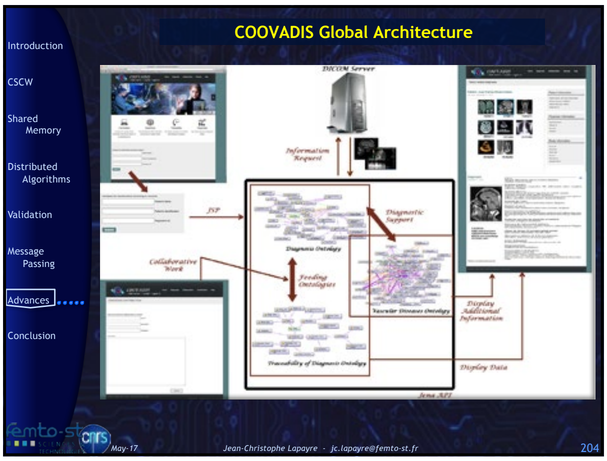

Ontologies
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Ontologies
Myocardial infarction
Cardiac infarction
heart attack
infarction
These terms represent the same pathology
They must be linked by the ontologies
Like that, everybody speaks about the same concept
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
199

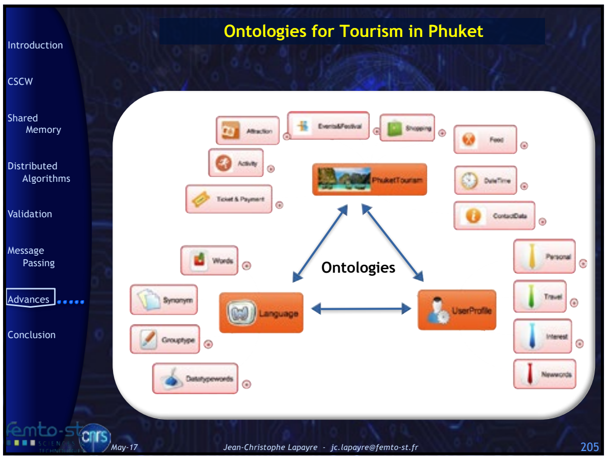

Ontologies for Traceability in Telemedicine
In medical systems, the most important objective is to identify, classify and protect the information using relational databases.
it allows to track the activities of medical procedures and treatments made by different departments and different professionals.
Concentration of Information
Diagnosis
Patient Information
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
200

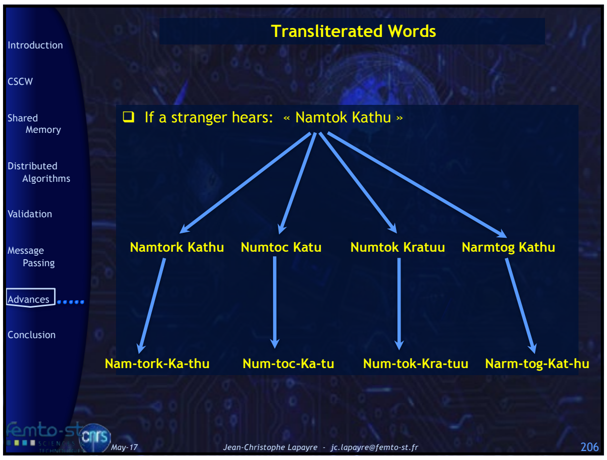

Ontologies: COOVADIS Platform
(Vascular Diseases)
Our objectives
Providing information sharing between physicians and improve the diagnosis in DICOM.
Traceability chain also includes the diagnosis using ontology technology, which is the result of physician interpretation of the initial study and further studies made by collaborative diagnosis community .
It is crucial that the professionals involved in collaborative diagnosis have total access to medical information and the up-to-date reports in the traceability chain.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
201


3 Ontologies as Support Tool for Physicians
Vascular Diseases Ontology
Symptoms and Causes
Disorders in Vascular System
Traceability of Diagnosis Ontology
Temporary Participation
Institution and Professional Involved
Diagnosis
Exams
Imagery
Clinical History
Diagnosis Ontology
These ontologies provide the required distributed information to the medical collaborative community for enabling a best patient monitoring and diagnosis support in COOVADIS.
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
202

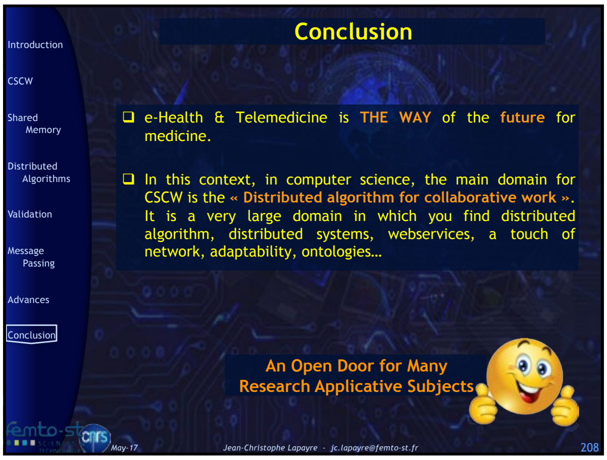

An Ontology: What Does It Look Like
Vascular Diseases Ontology: Defined with PROTÉGÉ Software
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
203

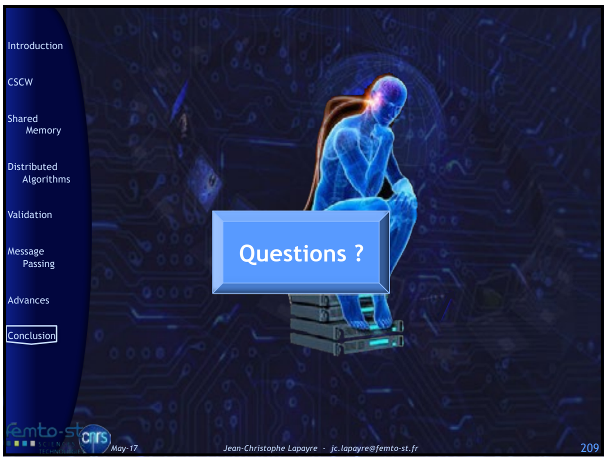

COOVADIS Global Architecture
DICOM Server
Information Request
JSP
Diagnostic Support
Collaborative Work
Feeding Ontologies
Display Additional Information
Display Data
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
204


Ontologies for Tourism in Phuket
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Ontologies
PhuketTourism
Language
UserProfile
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
205


Transliterated Words
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❑ If a stranger hears: « Namtok Kathu »
Namtork Kathu
Numtoc Katu
Numtok Kratuu
Narmtog Kathu
Nam-tork-Ka-thu
Num-toc-Ka-tu
Num-tok-Kra-tuu
Narm-tog-Kat-hu
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
206


Ontologies for Tourism in Phuket
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
A WebSite for Access
3 ontologies
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
207


Conclusion
Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
❑ e-Health & Telemedicine is THE WAY of the future for medicine.
❑ In this context, in computer science, the main domain for CSCW is the « Distributed algorithm for collaborative work ». It is a very large domain in which you find distributed algorithm, distributed systems, webservices, a touch of network, adaptability, ontologies…
An Open Door for Many
Research Applicative Subjects
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
208


Introduction
CSCW
Shared Memory
Distributed Algorithms
Validation
Message Passing
Advances
Conclusion
Questions ?
May-17
Jean-Christophe Lapayre - jc.lapayre@femto-st.fr
209