Verifying Safety of Fault-Tolerant Distributed Components

R. Ameur-Boulifa[(1)], R. Halalai[(2)], L. Henrio[(2)], E. Madelaine[(2)]

(1) Telecom-ParisTech

Sophia-Antipolis

(2) Oasis team

INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis

Initialy presented at FACS'2011, Oslo

Motivations

Programming asynchronous (component-based) applications is difficult, we must provide tools for analysing / debugging complex behaviours.

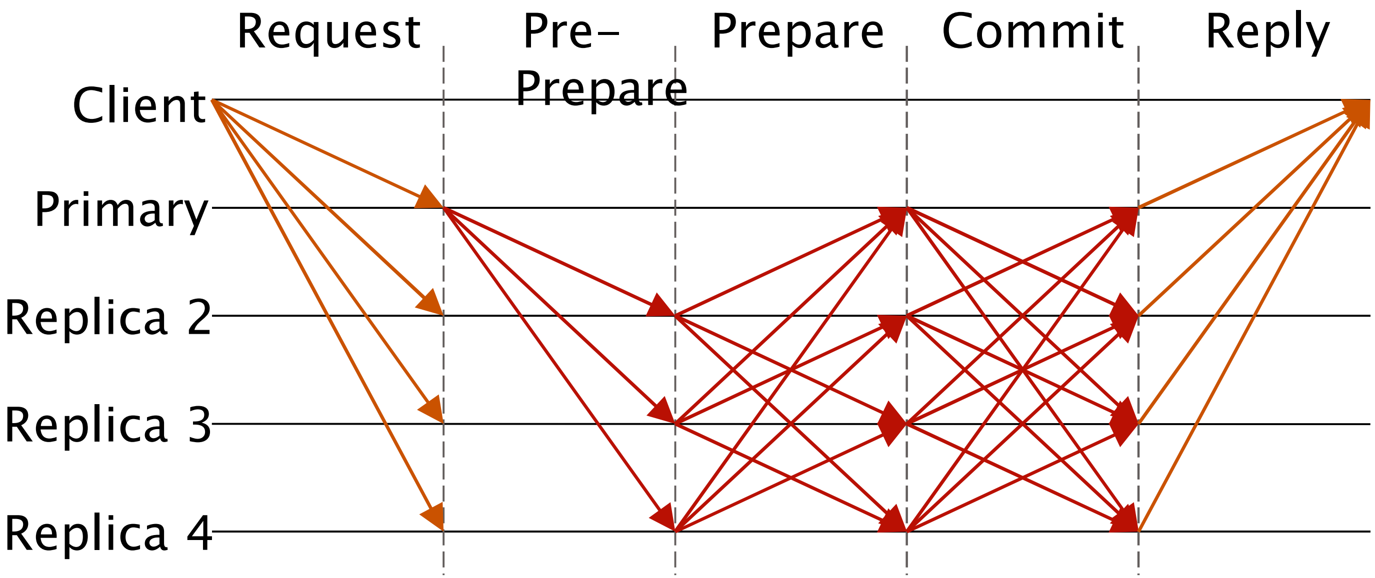
We want to provide a full behavioural semantics for Fractal/GCM components, including their advanced features: asynchronous request queues and future proxies, multicast interfaces.

“Compositional Model-checking can scale very far”

How far ?

Byzantine Fault Tolerant Systems

- **Byzantine hypothesis**:
  - "bad" guys can have any possible behaviour,
  - everybody can turn bad, but only up to a fixed % of the processes.



Byzantine Fault Tolerant Systems

- **Correction of BFT is difficult to prove *[see bibs in the paper]***
  … but is important in the context of large distributed infrastructures (e.g. P2P networks).

- **high complexity because of the behaviour of faulty processes, and asynchronous group communication.**

- **several advanced features of the GCM component model.**

Challenges

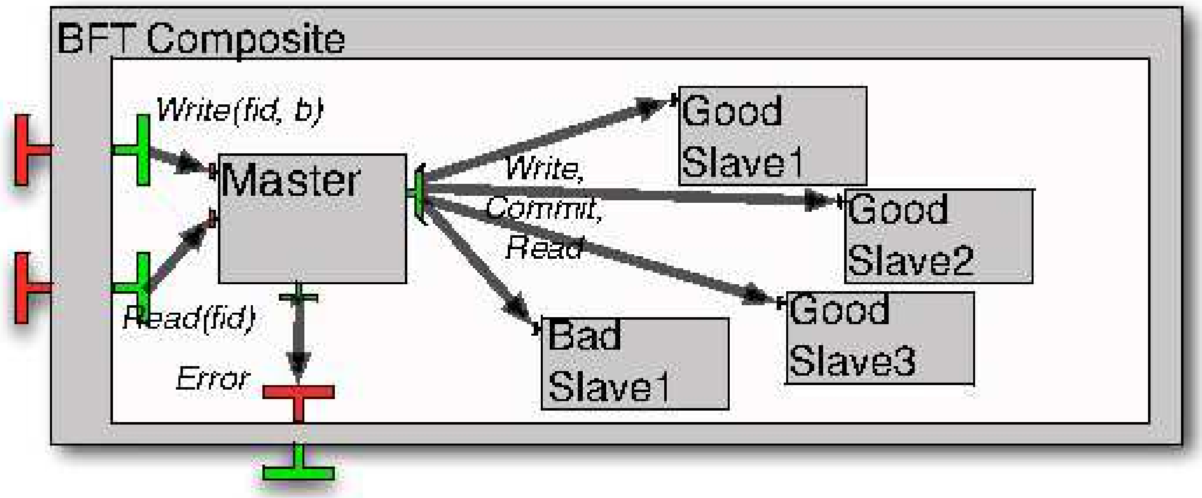
- **Scaling up**: are finite-state models able to tame complex, hierarchical, distributed systems ?
  - Compositionnality: hierarchical semantic model for hierarchical components
  - Bisimulations; context dependent minimization
- **Combining reduction techniques:**
  - Data abstraction + compositionality + distributed MC

Agenda

- **Use-case**
- **Formalisms and Semantics**
- **Use-case: state-space generation and model-checking**
- **Conclusion and Perspectives**

Use-case modeling in GCM

-1 composite component with 2 external services Read/Write.

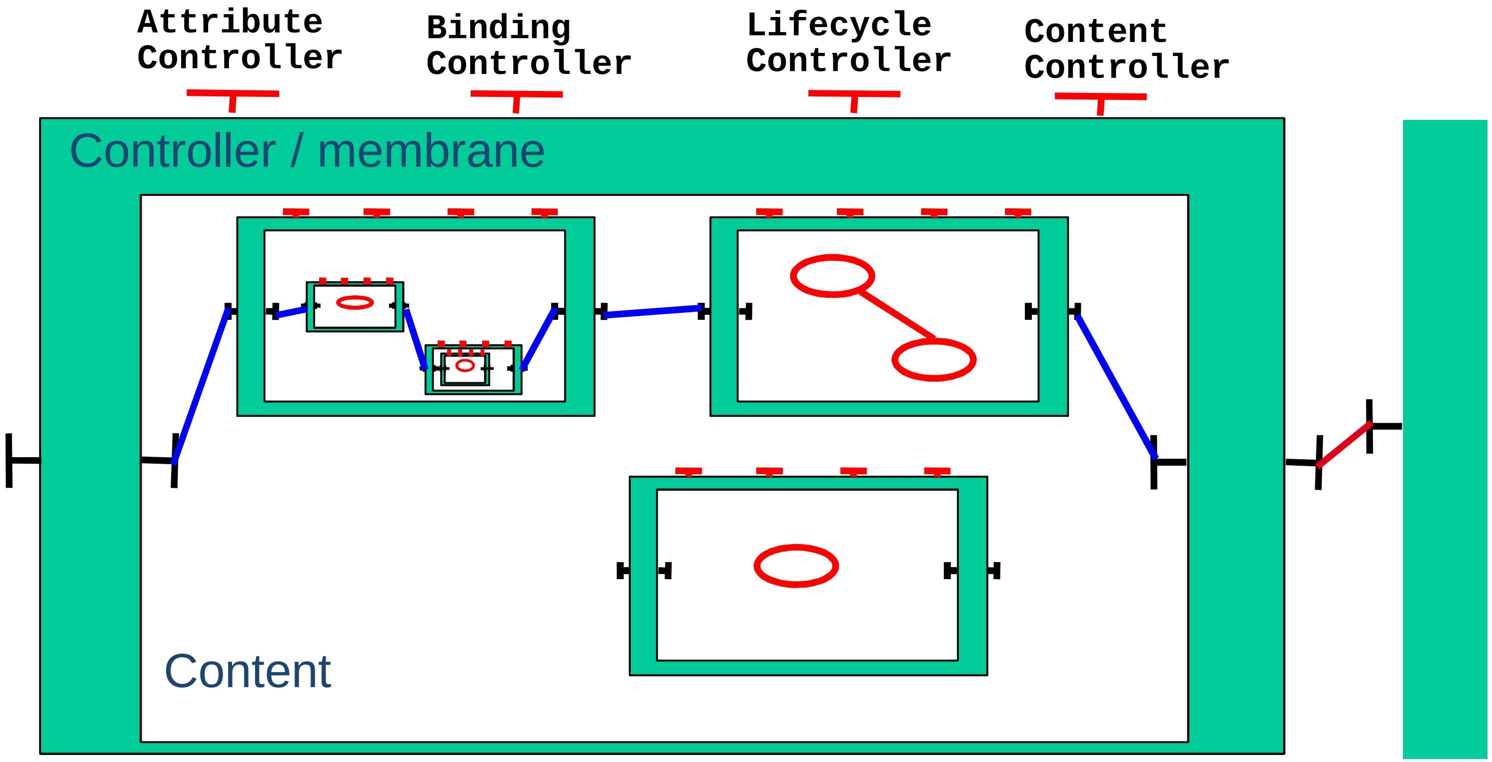


- The service requests are delegated to the Master.

- 1 multicast interface sending write/read/commit requests to all slaves.
- the salves reply asynchronously, the master only needs 2f+1 coherent answers to terminate

Fractal hierarchical model :

- Provided/Required Interfaces
- Hierarchy
- Separation of concern: functional / non-functional
- ADL
- Extensible



composites encapsulate primitives, which encapsulates code

Simplification hypothesis

1. **The master is reliable:** **this simplifies the 3-phases commit protocol, and avoid the consensus research phase.**
2. **The underlying middleware ensures safe communications**: **faulty components only respond to their own requests, and communication order is preserved.**
3. **To tolerate f faults** **we use 3f+1 slaves, and require 2f+1 agreeing answers, as in the usual BFT algorithms.**

Properties

Reachability(*):

1- The Read service can terminate

∀ fid:nat among {0...2}. ∃ b:bool. <true* . {!R_Read !fid !b}> true

2- Is the BFT hypothesis respected by the model ?

< true* . 'Error (NotBFT)'> true

Termination:

After receiving a Q_Write(f,x) request, it is (fairly) inevitable that the Write service terminates with a R_Write(f) answer, or an Error is raised.

Functional correctness:

After receiving a ?Q_Write(f1,x), and before the next ?Q_Write, a ?Q_Read requests raises a !R_Read(y) response, with y=x

(*) Model Checking Language (MCL), Mateescu et al, FM'08

Agenda

- Use-case
- **Formalisms and Semantics**
- State-space generation and model-checking
- Conclusion and Perspectives

Semantic Formalism : the pNet model

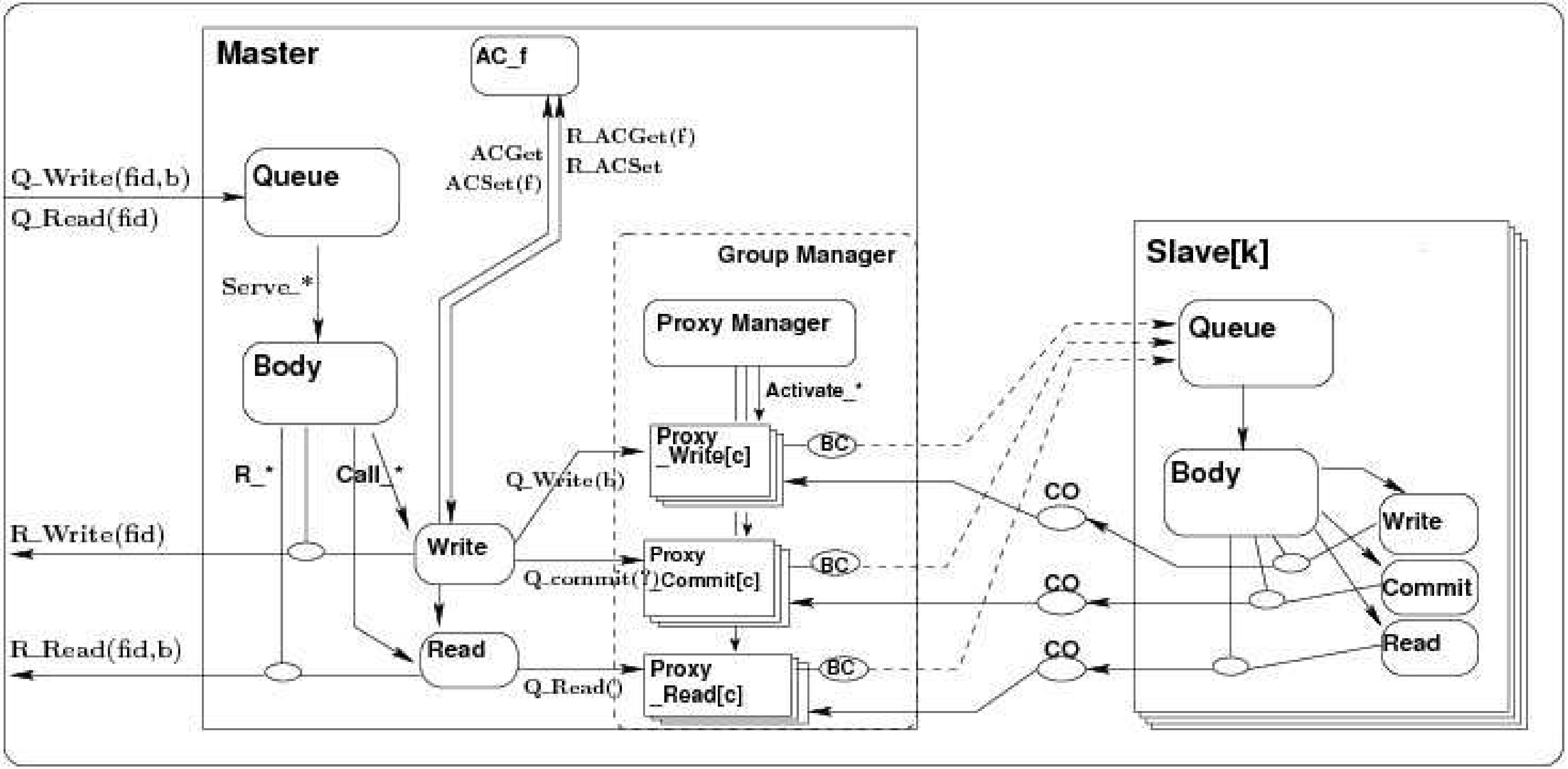
[Annals of Telecoms 2008]

- **LTS with explicit data handling (value-passing) with 1st order types**
- **Parallelism and hierarchy using extended synchronization vectors, with parameterized topology.**

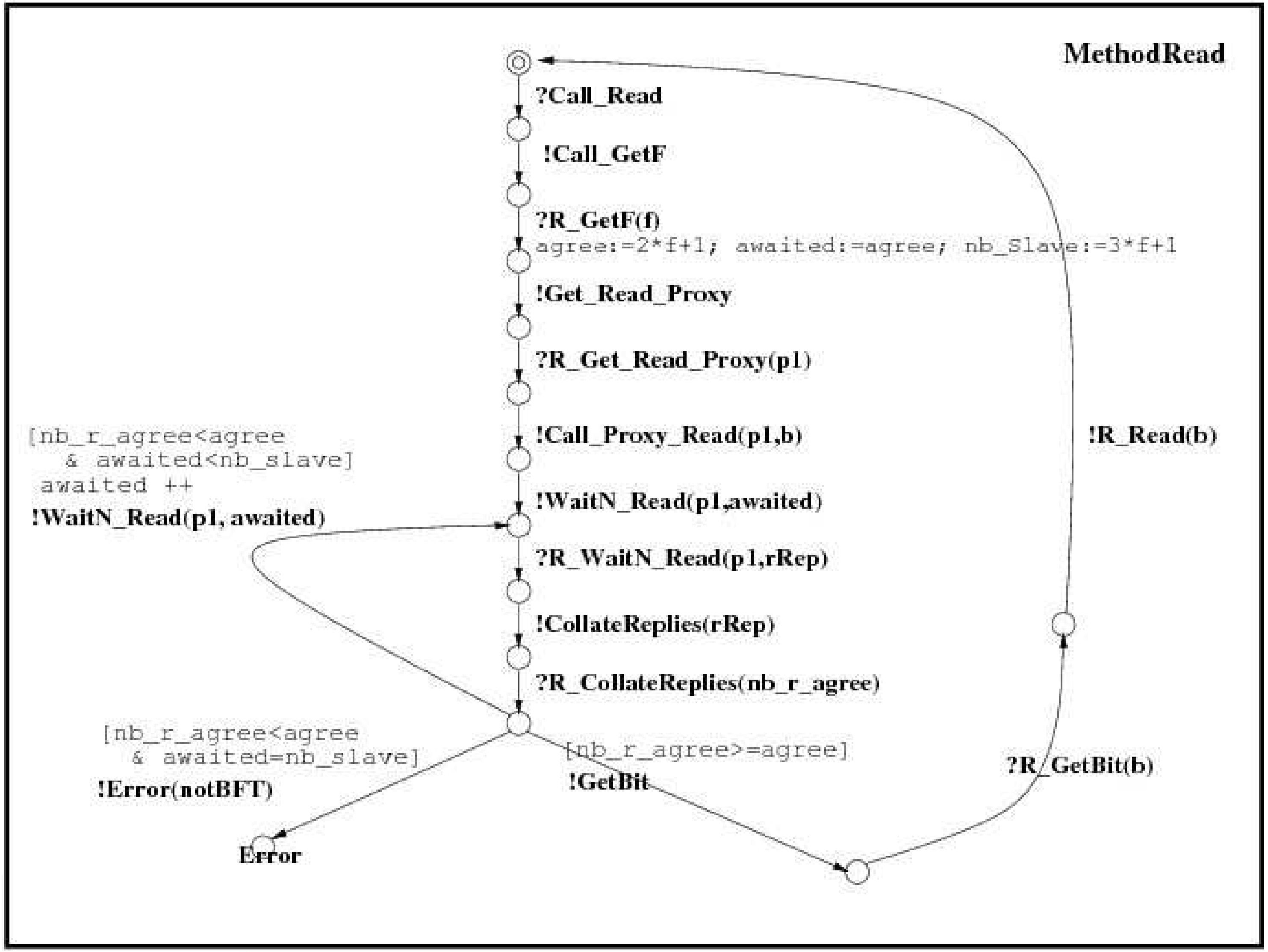
Compromise:

- **Flexible**: **accommodate a wide choice of communication / synchronization mechanisms**
- **Opened** **to convenient “abstractions” towards specific classes of decidable models (finite, regular, etc.)**

Full picture



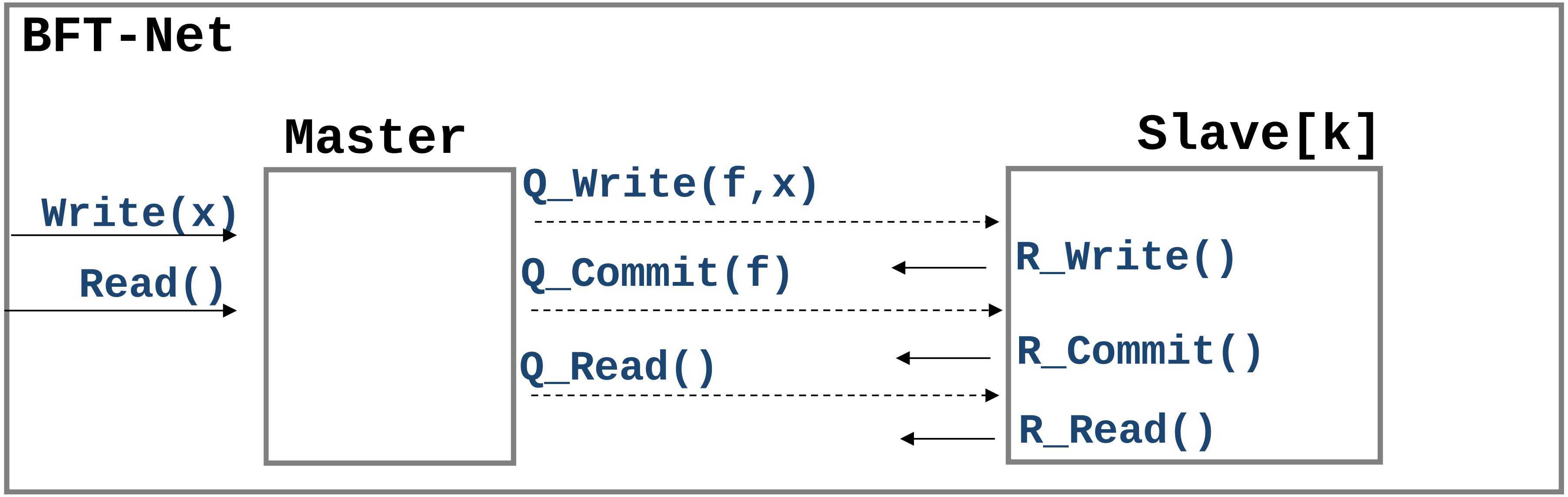
Building pNets (1) : parameterized LTSs



Labelled transition systems, with:

- **Value passing**
- **Local variables**
- **Guards….**

Building pNets (2) : generalized parallel operator



BFT-Net : < Master, Slave_1,…,Slave_n > k ∈ [1:n]

with **synchronisation vectors** :

<?Write(x), - , …, - > => ?Write(x)

<!Q_Write(f,x), ?Q_Write(f,x) , …, ?Q_Write(f,x) > => Q_Write(f,x)

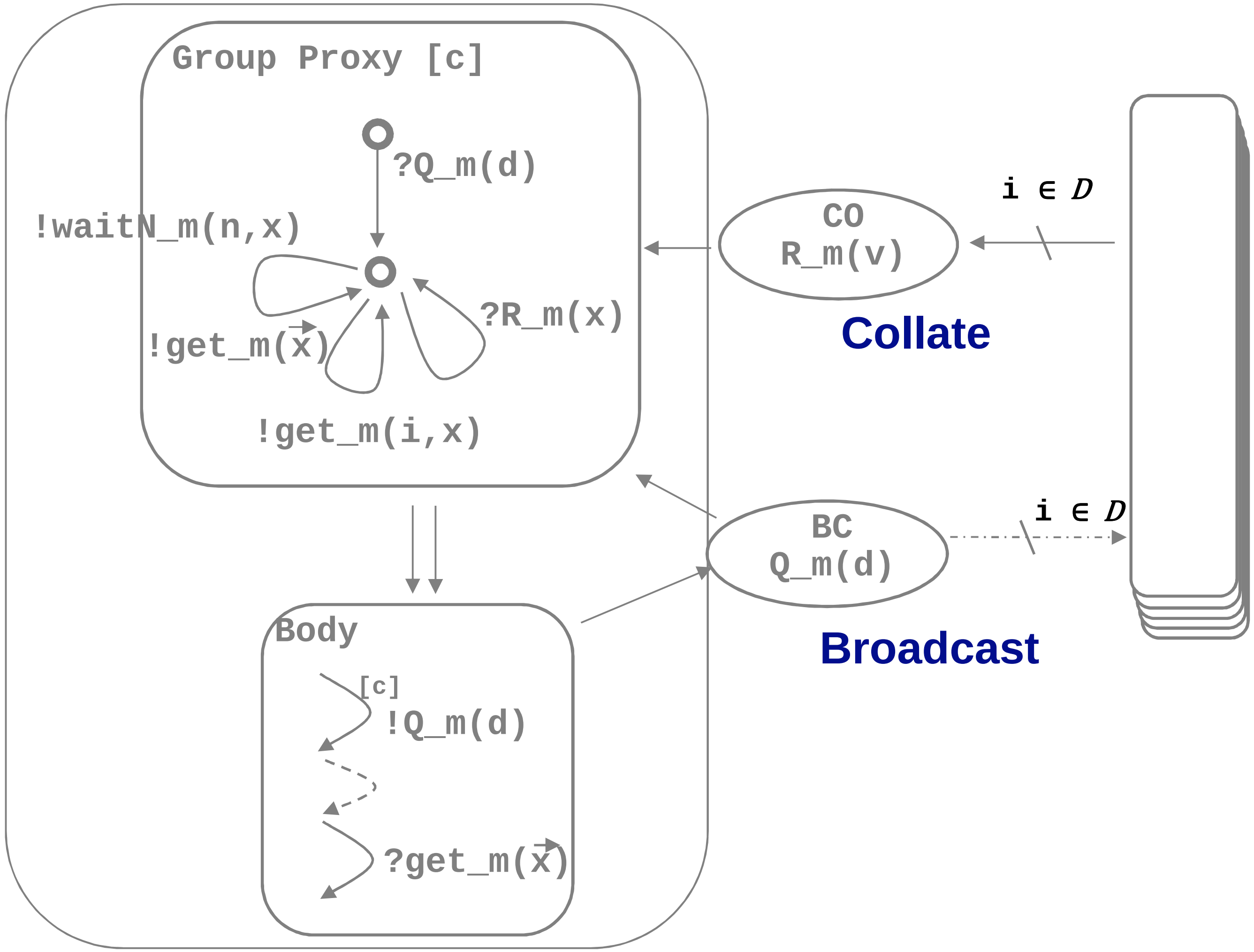
∀k <?R_Write(f,k), - , …, !R_Write(f), …, - > => R_Write(f,k)

Building pNet models (3)

Proxies for Asynchronous group requests

manage the return of results, with flexible policies:

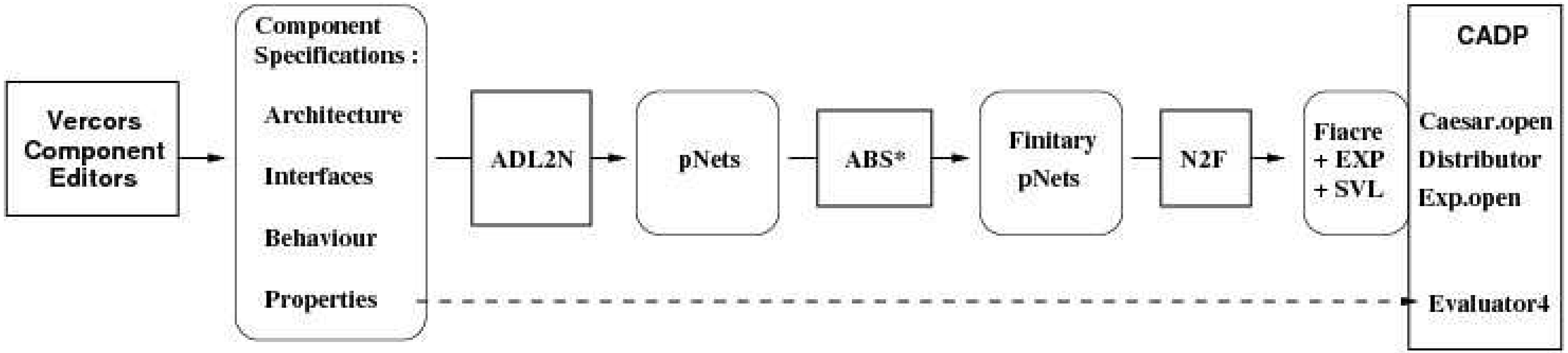
- Vector of results
- First N results
- Individual results



Agenda

- Challenges
- Formalisms and Semantics
- **Use-case: state-space generation and model-checking**
- Conclusion and Perspectives

Tool Architecture



Goal:

fully automatic chain

Current state of the platform:

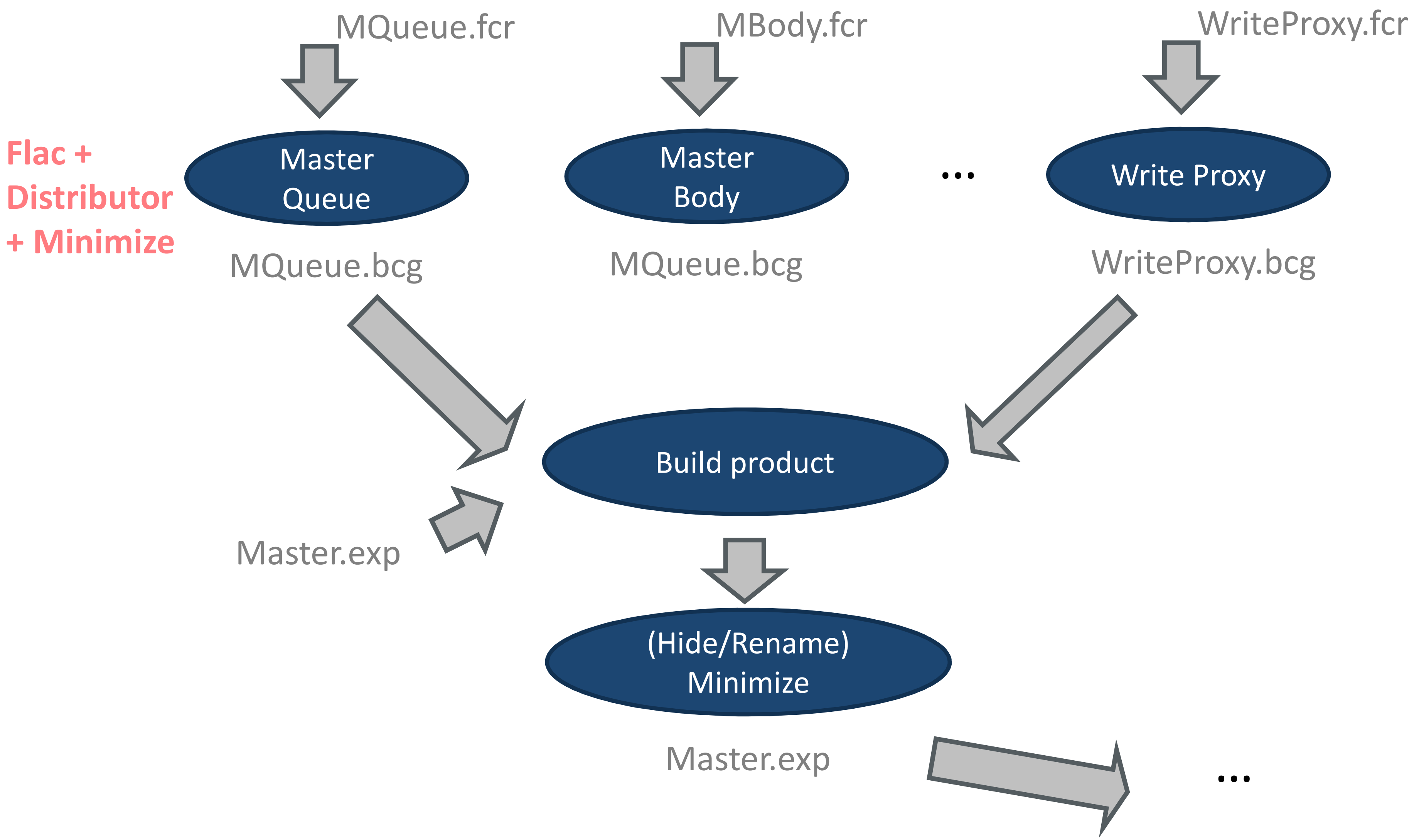
production of the CADP input formats only partially (~50%) available.

Generation of state-space

Taming state-space explosion:

- **Data abstraction (through abstract interpretation):**
  - **integers => small intervals**
  - **arrays ??? => open question.**
- **Partitioning and minimizing by bisimulation + context specification**
- **Distributed verification.**
  - **Only partially available (state-space generation, but no M.C.).**
  - **3 Tbytes of RAM ?**

State-space generation workflow



Distributed State generation

Abstract model:
f=1, (=> 4 slaves), |data|= 2, |proxies|=3*3, |client requests|=3

Master queue size = 2

~100 cores, max 300 GB RAM

System parts sizes (states/transitions):

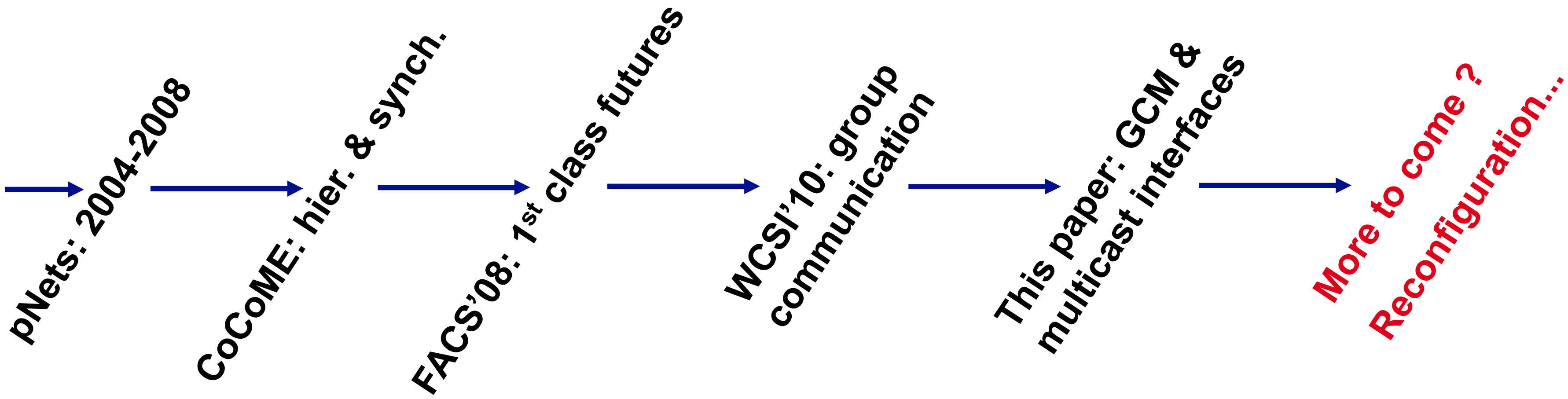
Queue	Largest intermediate	Master	Good Slave	Global
237/3189	524/3107	5M/103M	5936/61K	34K/164K

Time
59’

Estimated brute force state spaces :

10^{18}	6.10^{3}	~ 10^{32}

Conclusions



Contributions:

Semantics of GCM components with multicast interfaces.
Scaling-up : gained 2 orders of magnitude by a combination of:

- **data abstraction,**
- **compositional and contextual minimization,**
- **distributed state-space generation.**

Verification of the correctness of a simple BFT application.

Ongoing and Future Work

1. **Tooling**

2. **Verifying dynamic distributed systems (GCM + Reconfiguration):**
   - handle Life-cycle and Binding Controllers,
   - encode sub-component updates,
   - several orders of magnitude bigger.

3. **Support for distributed MC:**
   - scripting languages,
   - partitioning strategies

Open Questions

1. **More on data abstraction:**
   - **symmetry in useful data structures (intervals, arrays, …),**

2. **Context constraints:**
   - **ad-hoc correctness proofs (e.g. through proof obligations),**
   - **links with assume-guaranty approaches, with behavioural typing.**

3. **Tooling**:
   - **Assisted definition of (valid) abstractions.**
   - **Assisted definition of MC partitioning and strategies.**

Thank you

谢谢

Takk

Mulţumesc mult

Papers, Use-cases, and Tools at :

http://www-sop.inria.fr/oasis/Vercors

Partially Funded by ANR Blanc with Tsinghua Un. Bejing.

Active Objects (very short…)

-Runnable (mono-threaded) objects

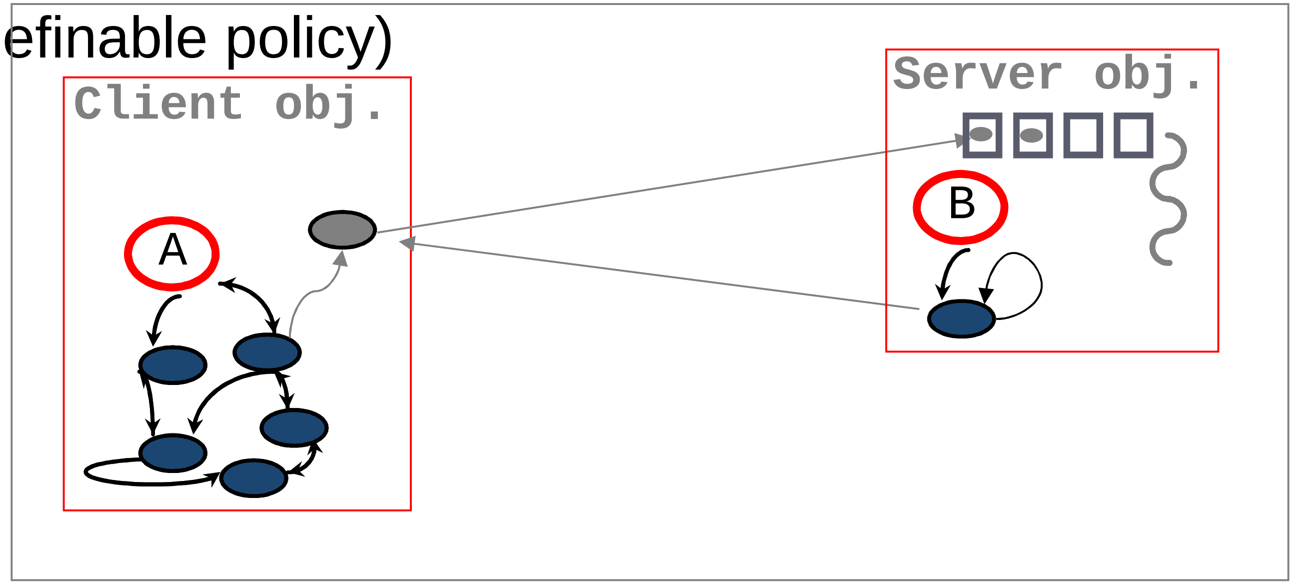
-Communicating by remote method call

-Asynchronous computation

-Request queues (user-definable policy)

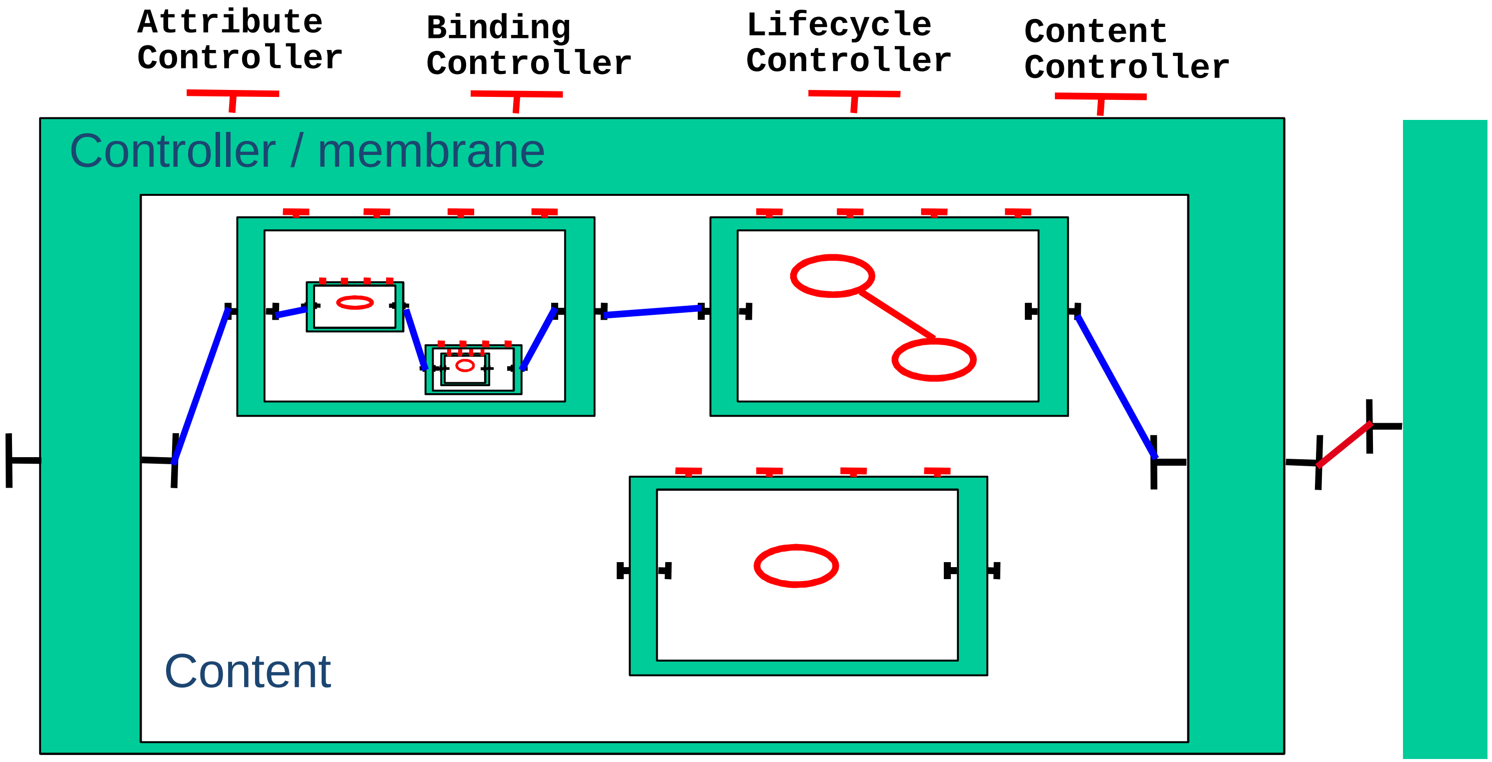
-No shared memory

-Futures



Fractal hierarchical model :

- Provided/Required Interfaces
- Hierarchy
- Separation of concern: functional / non-functional
- ADL
- Extensible



composites encapsulate primitives, which encapsulates code