Experiments with distributed Model-Checking of group-based applications

Eric Madelaine, Ludovic Henrio,

Rabéa Ameur-Boulifa, Raluca Halalai

Oasis team : INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis

SAFA Workshop, October 2010, Sophia-Antipolis

Université Nice Sophia Antipolis

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

INRIA

Motivations ?

We already have published case-studies of behavioural semantics and verification for distributed objects/components,

We wanted to explore:

- applicability of our approaches to “bigger” and more realistic cases,
- new tools for finite-state model-checking,
- execution on our large cloud infrastructure.

Characteristics of this study:

- Asynchronous (but bounded) request queues
- Parameterized system (value passing _and_ topology)
- Group communication

This presentation is an extension of:

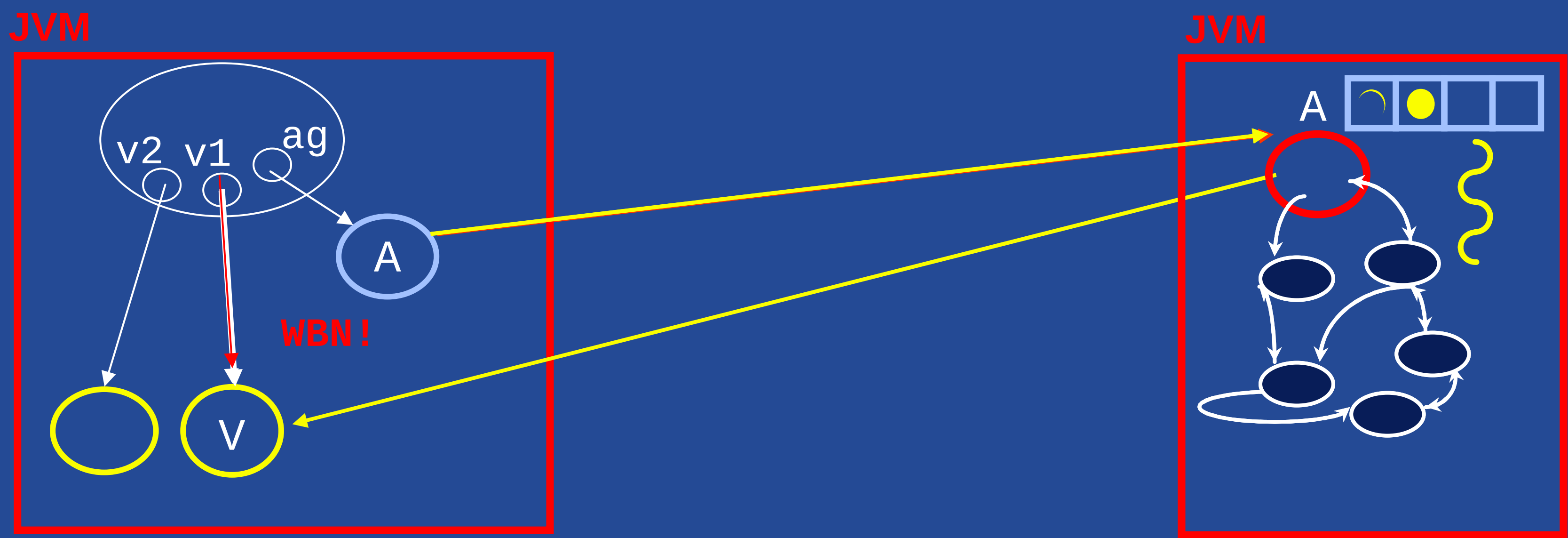
Behavioural Models for Group Communications, WCSI, Malaga, 2010 (EPTCS)

Agenda

- Background
  - Active Objects, Groups, the VerCors platform
- Models for groups
  - The Case-study
  - Behavioural semantics of broadcast messages and asynchronous proxies
- State generation and Verification
  - State space generation: sequential / distributed / hierarchical
  - Proving properties
- Conclusion & Perspectives

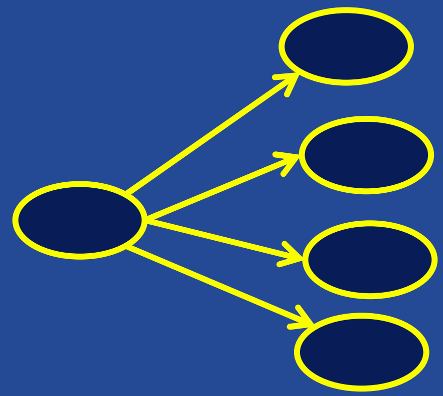
ProActive : Active objects

```
A ag = newActive ("A", […], VirtualNode)
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```



Wait-By-Necessity is a Dataflow Synchronization

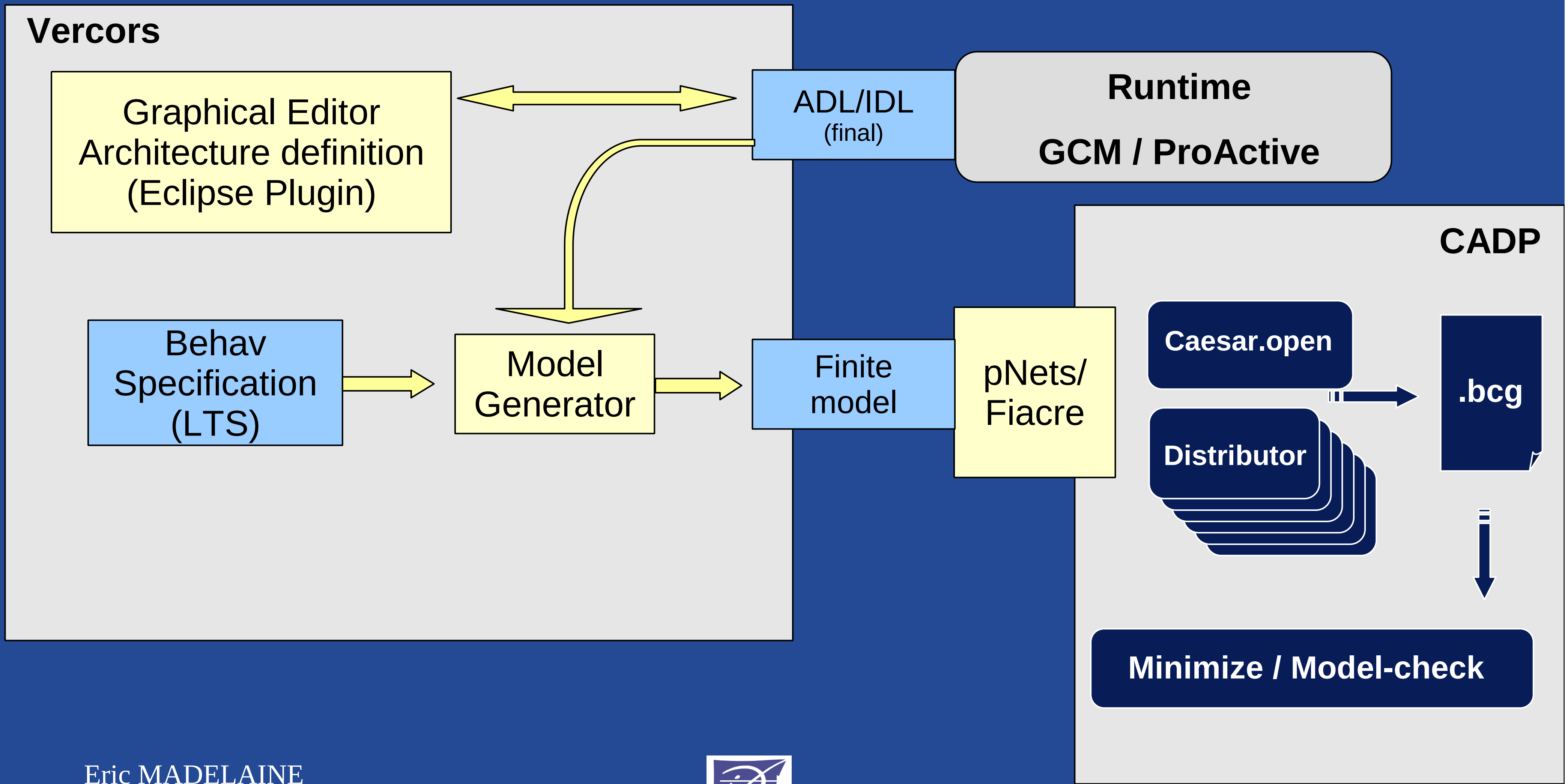
Groups



- One-to-many communication
  - A single instruction for many communications
  - Allows optimisations and specific synchronisations

  ➔ A convenient programming abstraction
- Specially useful for SPMD programs, but also for most of distributed applications
- Several data distribution policies are possible, e.g.:
  - Scatter
  - Broadcast

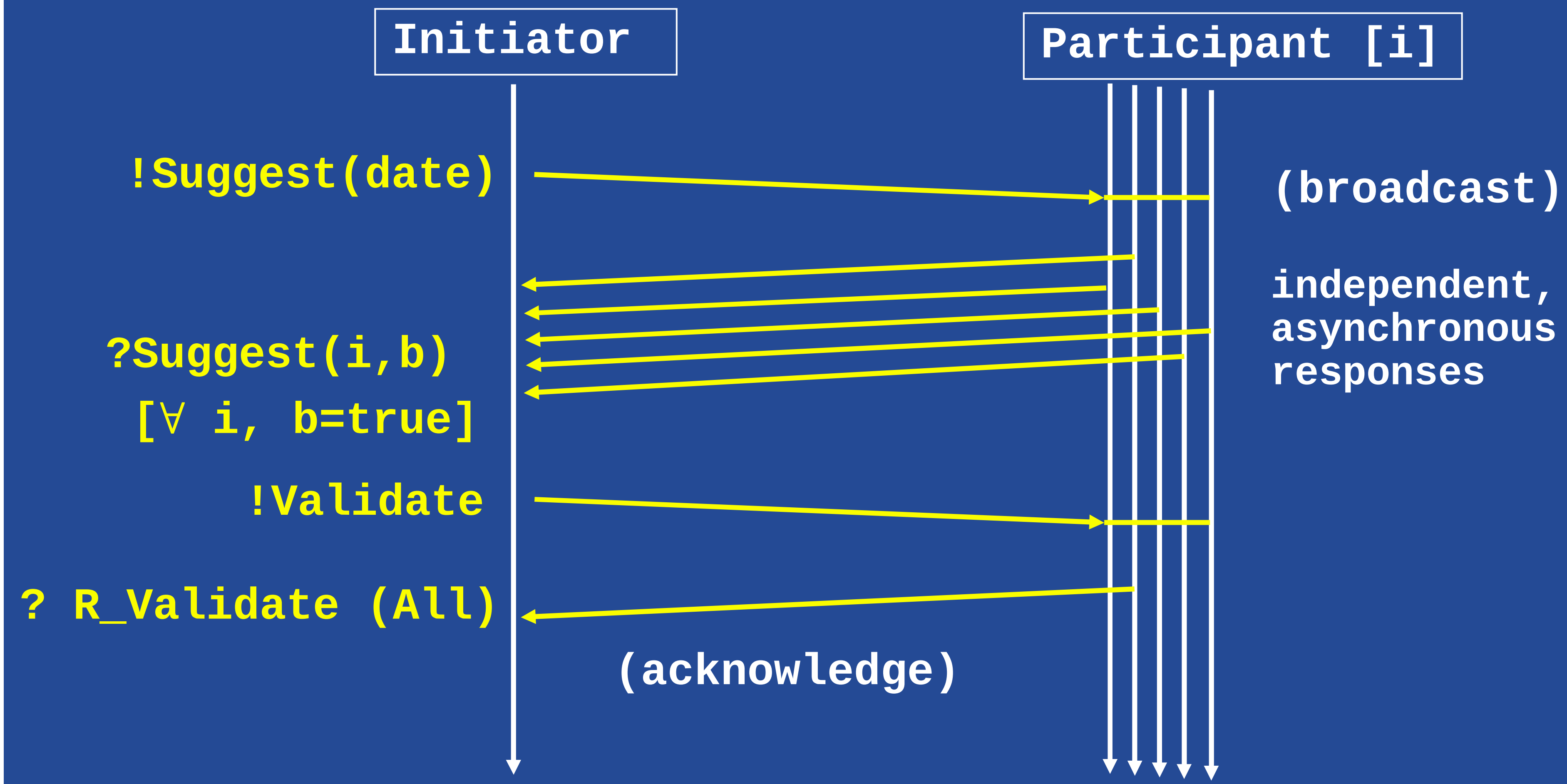
The Vercors Specification and Verification Platform (current prototypes)



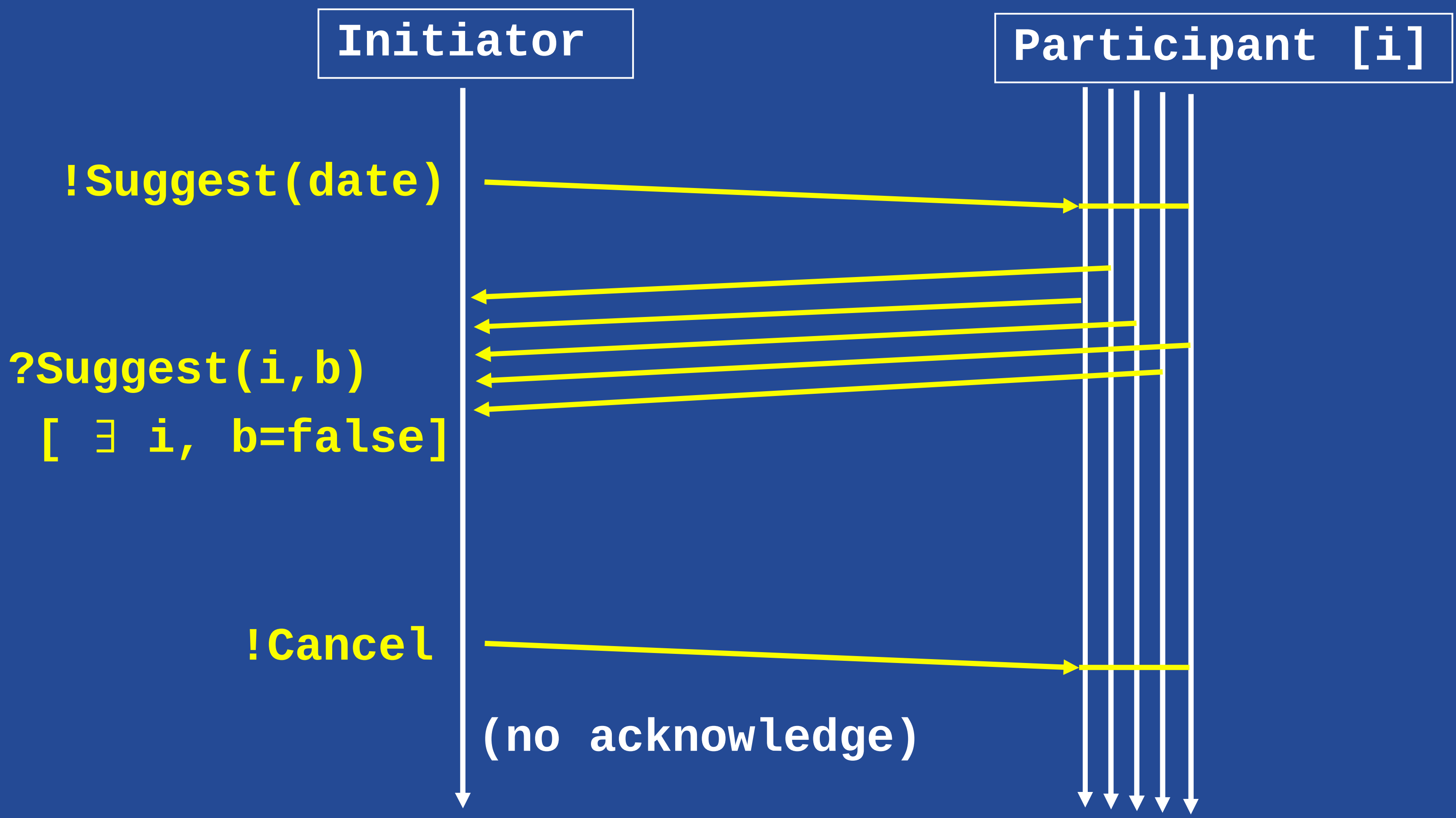
Agenda

- Background
  - Active Objects, Groups, the VerCors platform
- **Models for groups**
  - **The Case-study**
  - **Behavioural semantics of broadcast messages and asynchronous proxies**
- State generation and Verification
  - State space generation: sequential / distributed / hierarchical
  - Proving properties
- Conclusion & Perspectives

Running Example :
« rendez-vous agreement » (1)



Running Example :
« rendez-vous agreement » (2)



Running Example :
« rendez-vous agreement »

Properties ?

- Absence of deadlocks
- Progress or termination (reachability)
- Inevitability
- Boundedness (of request queues)

Agenda

- Background
  - Active Objects, Groups, the VerCors platform
- **Models for groups**
  - The Case-study
  - **Behavioural semantics of broadcast messages and asynchronous proxies**
- State generation and Verification
  - State space generation: sequential / distributed / hierarchical
  - Proving properties
- Conclusion & Perspectives

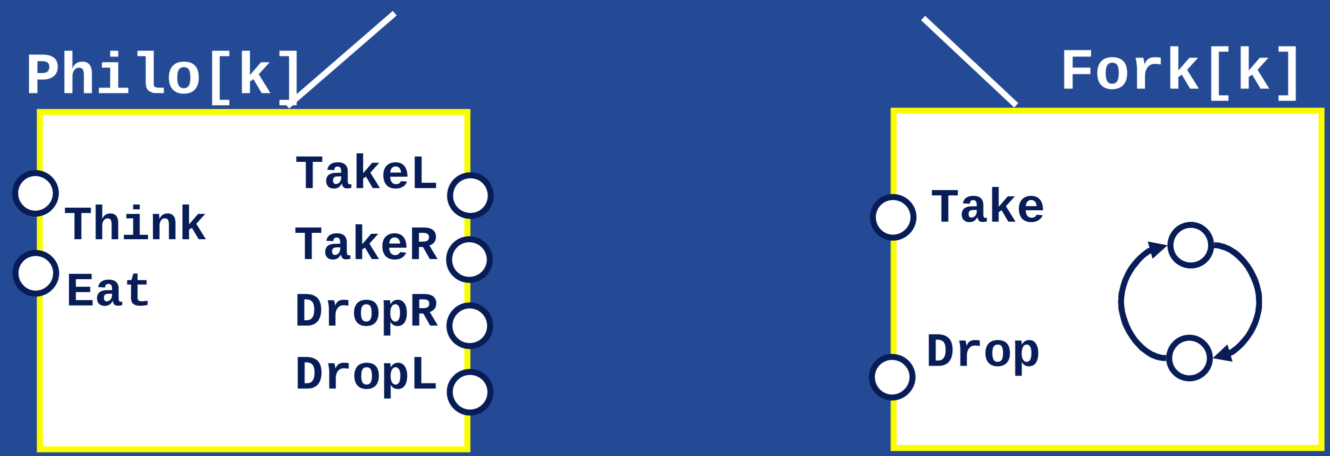
Parameterized Networks of Synchronised Automata (pNets)

We have used them to formalize the behavioural semantics of:

- Active Objects (Forte'04)
- Objects with first class futures (Facs'08)
- Fractal components, distributed components, reconfiguration (Annals of Telecom'09)

synch vectors : <Think(k), Think $_{Philo[k]}$ >

<TakeL(k), TakeL $_{Philo[k]}$, Take $_{Fork[k-1]}$ >, …

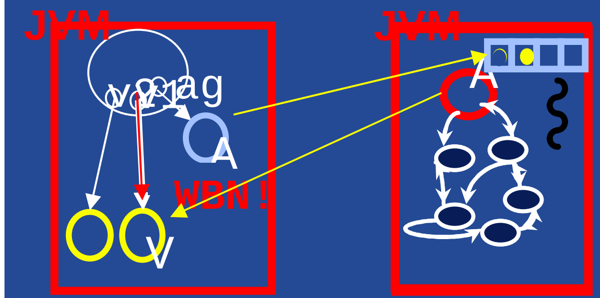


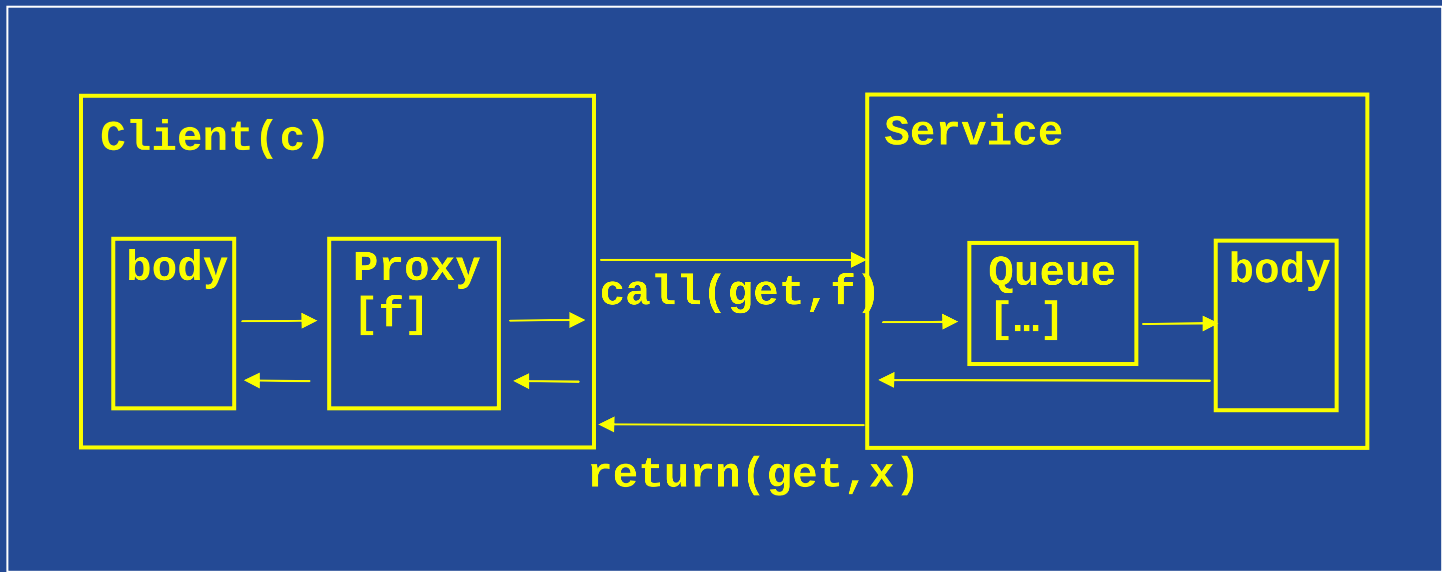
Building pNet models (1)

Nets for **Active objects communication** schema :

From the set of public methods, and their signature, build :

- The (parameterized) action algebra
- The structure of the future proxies and the request queue
- One synchronisation vector per exchanged message.



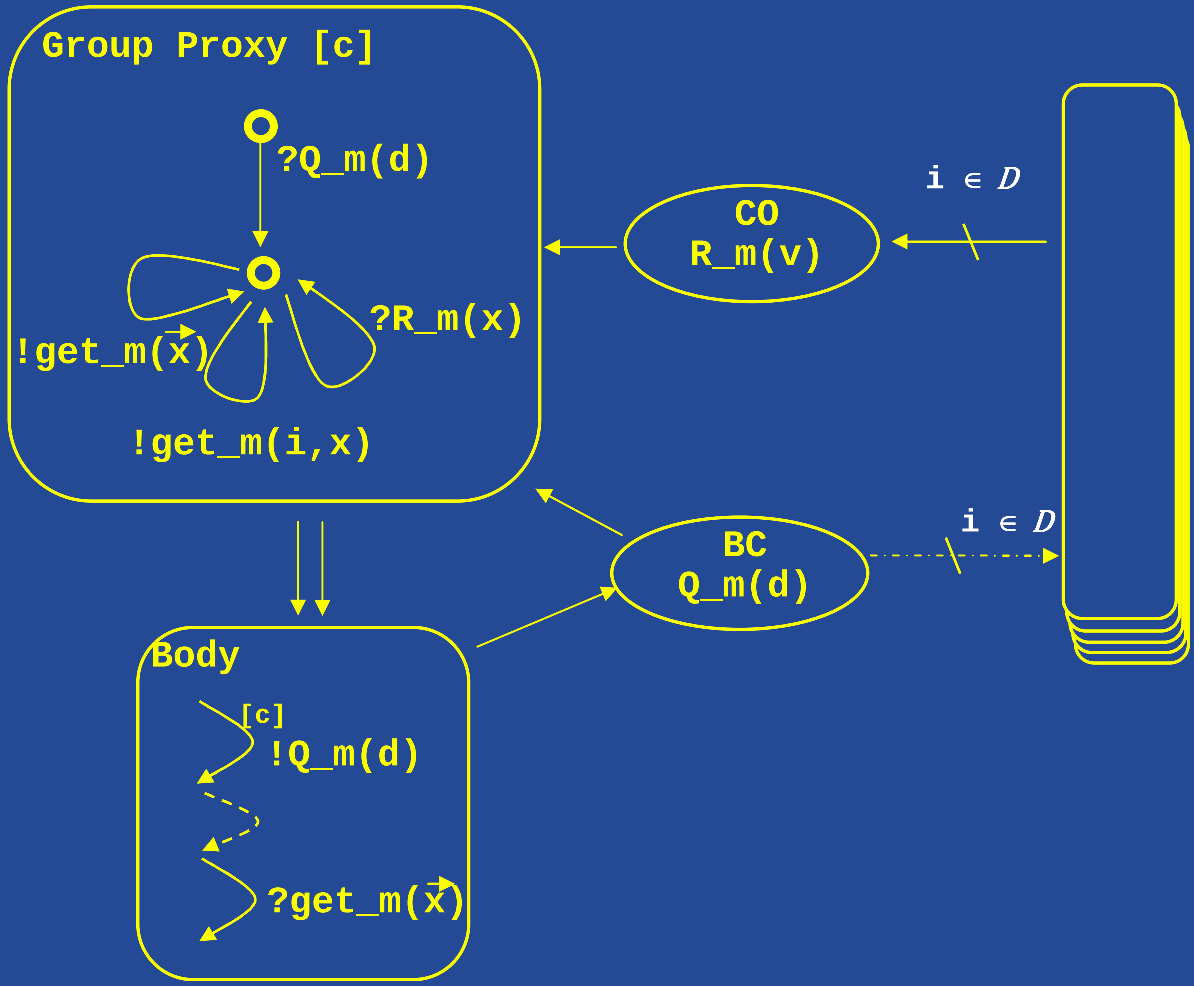


Building pNet models (2)

Proxies for Asynchronous requests

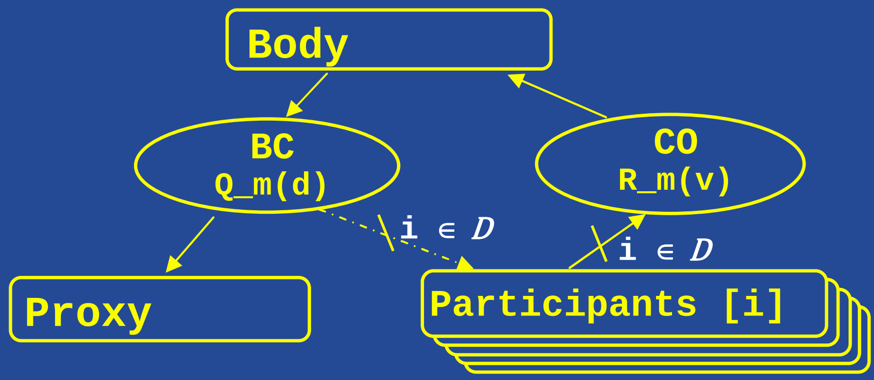
Manages the return of results, with flexible policies:

- Vector of results
- First N results
- Individual results



Building pNet models (2)

Group communication :



BC= Broadcast

One single synch vector for all participants

$<Q\_m(d), !Q\_m(d)_{Body}, ?Q\_m(d)_{Proxy}, ?Q\_m(d)_{P[1]}, \ldots, ?Q\_m(d)_{P[n]}>$

CO= Asynchronous Collection

One synch vector for each participant in the group

$<R\_m(x), ?R\_m(d)_{Body}, *, *, \ldots, !R\_m(x)_{P[i]}, \ldots, *>$

$<R\_m(x), ?R\_m(d)_{Body}, *, !R\_m(x)_{P[i]}, *, \ldots, *>$

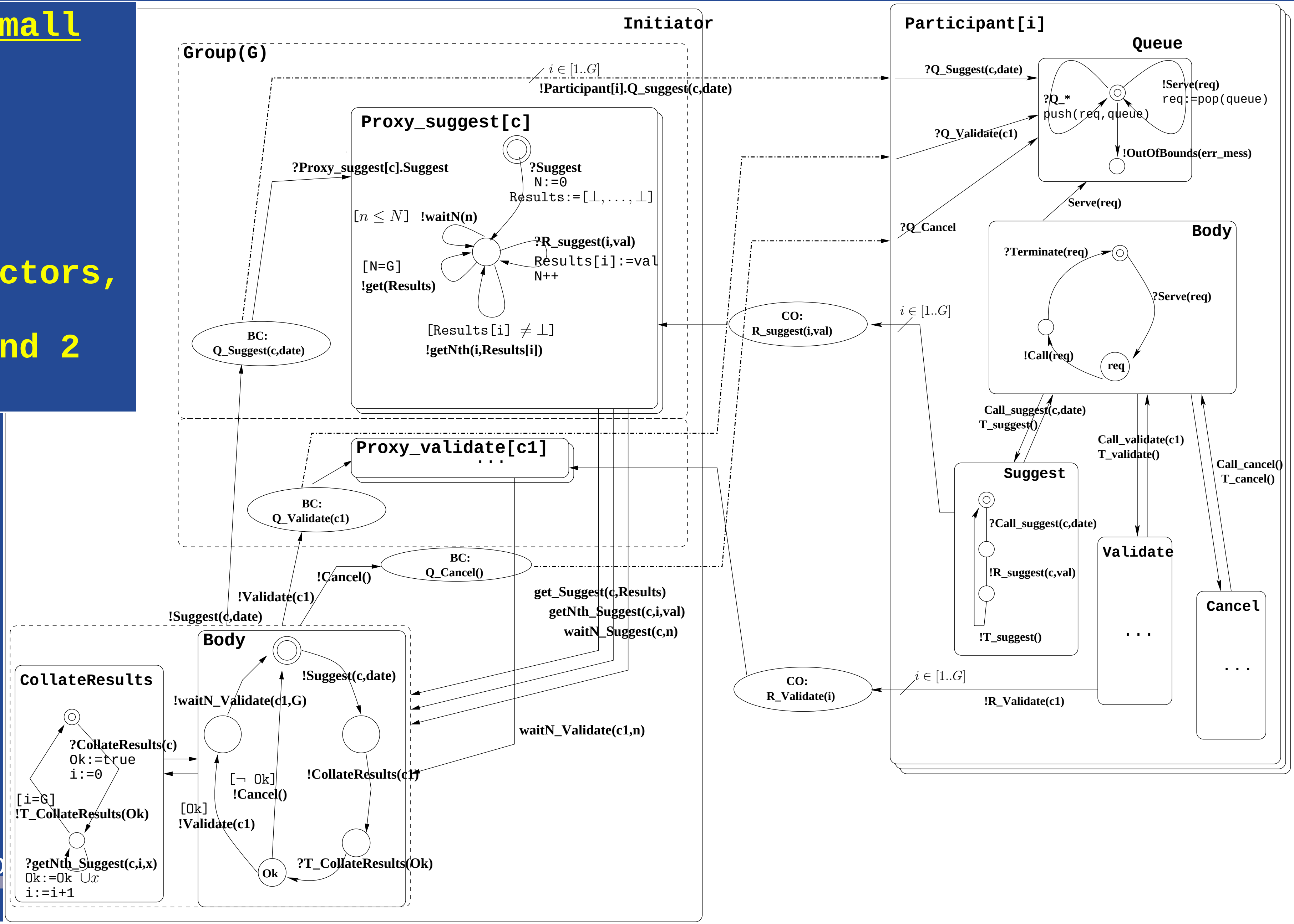
Generated Model: the full picture

This is a small system:

10 pLTS

7 parameter variabless

19 synch vectors, including 3 broadcast and 2 collectors.



Agenda

- Background
  - Active Objects, Groups, the VerCors platform
- Models for groups
  - The Case-study
  - Behavioural semantics of broadcast messages and asynchronous proxies
- **State generation and Verification**
  - **State space generation: sequential / distributed / hierarchical**
  - **Proving properties**
- Conclusion & Perspectives

State generation 1: classical

3 participants
Data ∈ { d1,d2 }
Res ∈Bool
15 visible labels

Machine:

Fedora 10, 4Go RAM

2 dual-core proc@2,4Ghz

	Brute force	Minimized	Total time
Single participant	1 801 / 5 338	90 / 376	8 "
Initiator	3 163 / 152 081	54 / 1 489	11 "
Full system, queue of length 2	170 K / 1 646 K	458 / 1 284	406 "
Group of 3 participants	> 10^11 states	-	-

- With no hierarchical minimization: the generation of a stand-alone group of 3 participants would be impossible
- It is essential to build sub-systems in the correct context (= behavioural contract) => e.g. Projector tool of CADP.

State generation 2: distributed

Principles:

- State space partitioned on a cluster by a static hash function. No shared memory.
- The state space is merged before other tools (minimization, model-checking) can be applied. Distributed MC is planned in future versions of CADP.

On the fly partial order reduction available:

- Tau-compression (collapsing tau-chains)
- Tau-confluence (selecting only representatives of confluent-sets)

State generation 2: distributed

	generation	Brute force	Total time
Full system with 3 participants (8x4 cores)	**Brute force** **Tau-compression** **Tau-confluence**	**170 K / 1 646 K** **170 K / 607 K** **5 K / 14 K**	**6'45** **11'48** **30'**
Group of 2 participants (15x8 cores)	**Brute force** **Tau-confluence**	**13 M / 48 M** **392 K / 1 354 K**	**11'32** **19h 10'55**
Group of 3 participants	**Brute force** **Tau-confluence**	**(*estimated 125 G states*)** **Out of memory during local computation**	

- Distributed state generation has a (fixed) overhead, but allow for very large RAM configurations. The bottleneck is the merge phase.
- On-the-fly partial-order techniques may help to save memory space, at a high price. It may also fail…

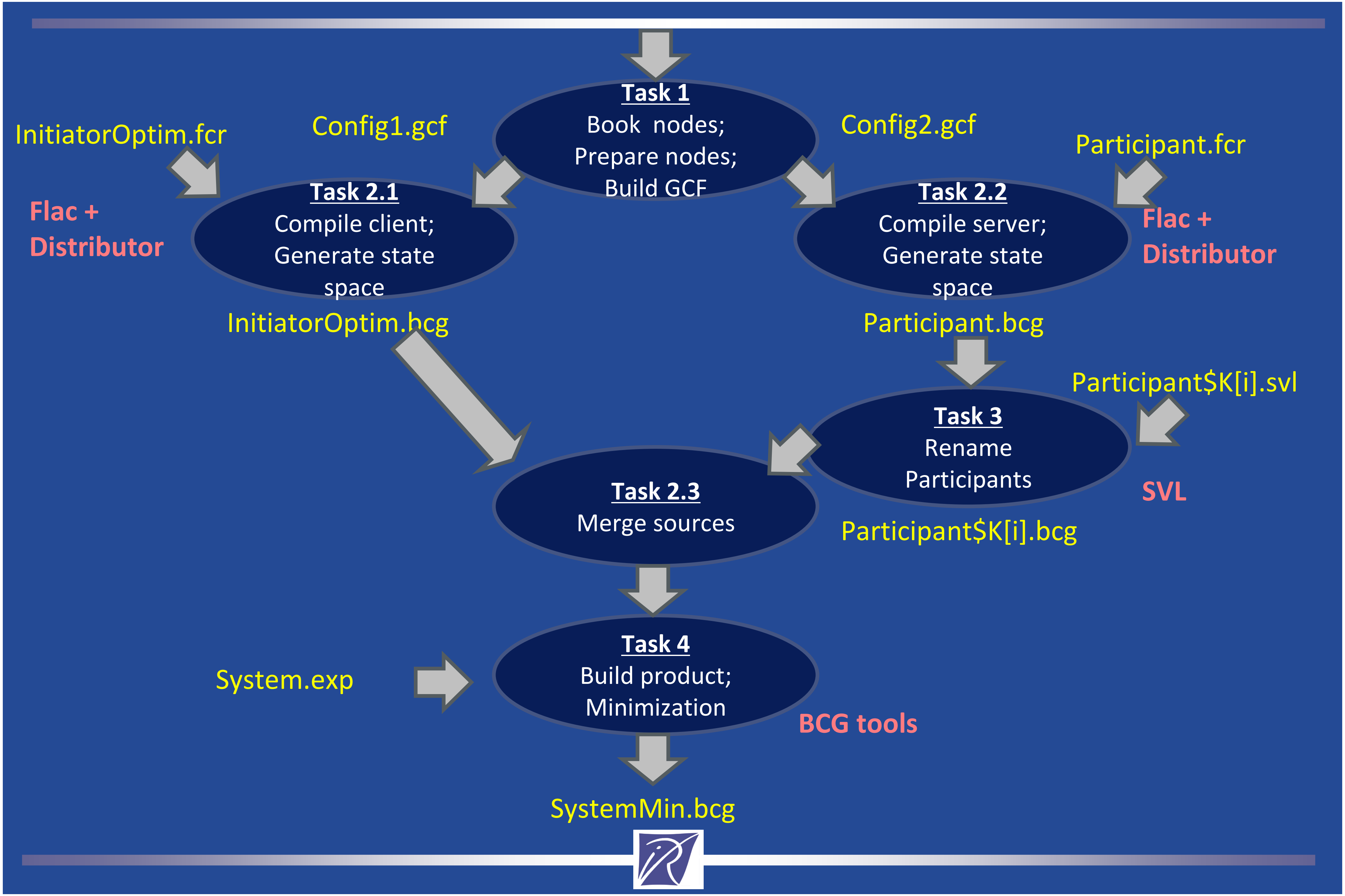
State generation 3: hierarchical

Classical compositional state generation:

Split the application into smaller pieces, minimize each with (branching) bisimulation before combining them.

Distributed verification architecture:

Define the verification activities as a workflow, and use a generic scheduler on the cloud infrastructure. Some of the workflow nodes are multinode (distributed) tasks.


Task 1
Book nodes;
Prepare nodes;
Build GCF
InitiatorOptim.fcr
Config1.gcf
Config2.gcf
Participant.fcr
Flac +
Distributor
Task 2.1
Compile client;
Generate state
space
Task 2.2
Compile server;
Generate state
space
Flac +
Distributor
InitiatorOptim.bcg
Participant.bcg
Participant$K[i].svl
Task 3
Rename
Participants
SVL
Task 2.3
Merge sources
Participant$K[i].bcg
System.exp
Task 4
Build product;
Minimization
BCG tools
SystemMin.bcg

State generation 3: hierarchical

Classical compositional state generation:

Split the application into smaller pieces, minimize each with (branching) bisimulation before combining them.

⇒The biggest intermediate structure has ~ 3000 states before reduction.

⇒A group of 3 (reduced) participants would be 90^3 = 800 000 states.

Distributed verification architecture:

Define the verification activities as a workflow, and use a generic scheduler on the cloud infrastructure. Some of the workflow nodes are multinode (distributed) tasks.

=> Open questions: formalism and tool support to specify

- the structural splitting
- the mapping to verification tasks.

Proving properties

- These experiments have been done while developing the encoding, so we had real opportunities to find bugs (and we did)

Properties proved:

- *Reachability and progress:*

Regular μ-calculus

$$< True * .T\ CollateResult\ (false) > True$$

$$< True * .R\ suggest\ (i,b) > True$$

- *Inevitability:*

Specification patterns

$$After\ \ !Q\_Suggest(id)\ \ Eventually\ \ !Q\_Cancel(.) \vee !Q\_validate(.)$$

- *Boundedness:*

$< True * .Error > True$ *(with queues of length 1)*

$[True * .Error]\ False$ *(with queues of length 2)*

Conclusions

We have presented:

- A behavioural semantics for group-based distributed applications, based on the pNets formalism, allowing for finite-state model-checking in the CADP platform
- Practical results in term of state generation (space and time) with various strategies, including distributed state-generation.
- Hierarchical state space generation/minimization using a generic cloud infrastructure.

Perspectives:

- Automatisation of the pNet encoding in our VerCors platform, with the challenge of delivering these tools to non-specialists
- Extension to distributed components, including reconfiguration.
- (Much) bigger experiments, …

Papers, Use-cases and Tools, Position Offers at :

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

Bonuses

- The CADP Toolset
- pNets: some elements of formal definitions
- Fiacre encoding
- Discussion
- PacaGrid infrastructure

Verification Tools

CADP toolset **(INRIA Rhones-Alpes, VASY team)**

- Generic Front-end
  - (Lotos, BCG, Sync-vectors, Fiacre)
  - symbolic simulator, graph exploration, etc.
- Distributed Model generation
  - Up to billions of states
  - On-the-fly, Tau-reduction, Constrained…
- Evaluator model-checker
  - Deadlock search / Regular μ-calculus
- Bisimulation ckecking, minimization

pNets and Nets : operators

- pNets are **generalized synchronisation operators** at the semantic level. They address: multiway synchronisation, parameterized topologies, and dynamic topologies.

Definitions:

- A **System** is a tree-like structure with pNets at nodes and pLTS at leaves
- **Data Abstraction**: given a countable (resp, finite) domain for each parameter of a system, its **instantiation** is a countable (resp. finite) system.
  - Value Passing case : Preservation of safety and liveness properties [Cleaveland & Riely 93]
  - Parameterized topologies : no similar result in general.

The FIACRE intermediate format

⇒ Low level semantic format, from the Fiacre project, and OpenEmbedd platform

```
process Queue [ Q_Suggest: in data,
    Q_Validate: in data2, Q_Cancel: none, ... ]
is
  states S_empty, S1, S2, ...
  var x:data, y:data2, ...

from S_empty
select
   Q_Suggest?x; to S1
[]
   Q_Validate?y; to S2
...
end
```

```
component System [Q_Suggest: data, ...]
is
  port R_Validate0, ...: indexG

par Q_Suggest, ... in
   R_Suggest0, R_Validate0, ... -> Initiator
           [Q_Suggest, R_Suggest0, R_Validate0, ...]
||
   R_Suggest0, R_Validate0 -> Participant0
           [Q_Suggest, R_Suggest0,... ]
||
...  end
```

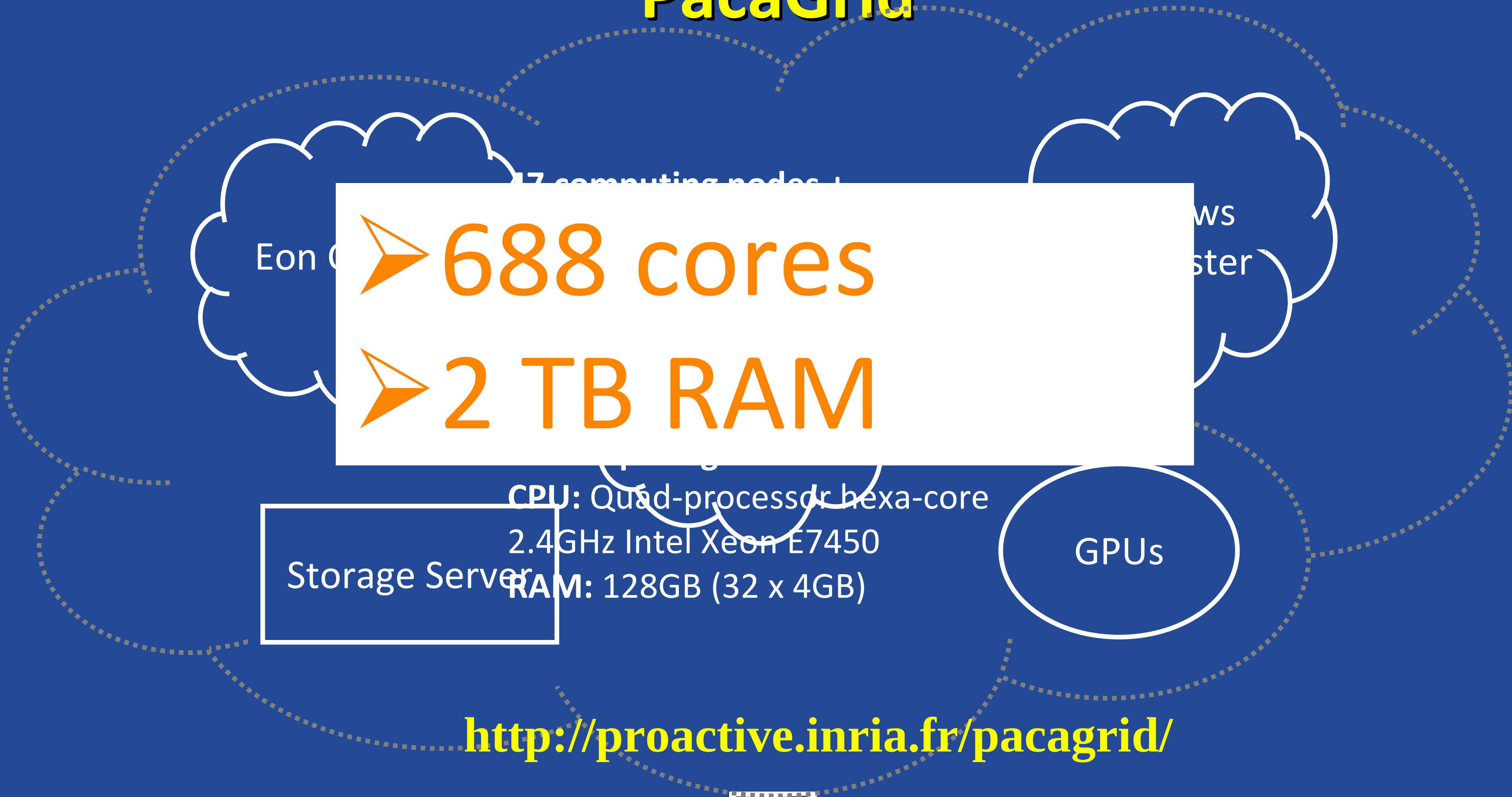
Discussion

1. Why do we need a request queue of length 2 ?

2. Optimal model generation means combining many techniques:
   - Use context (contract) to generate the models of basic sub-systems
   - Abstract (data, visible events) in an optimal way
   - Build / minimized hierarchically
   - Need for a lot of intelligence in the scripting language (SVL)

3. Flexibility, monitoring, control of the distributed tools
   - Very few distributed platforms (CADP, DiViNE, U.Twente LTS-min)
   - Move to standardized grid/cloud infrastructures?

PacaGrid



- 688 cores
- 2 TB RAM

http://proactive.inria.fr/pacagrid/