Multi-active Objects

Ludovic Henrio[1], Fabrice Huet[1], Zsolt István[2], Gheorghe Sebestyén[2]

[1]INRIA Sophia-Antipolis, CNRS, Univ Nice sophia Antipolis [2]Technical University of Cluj-Napoca

SAFA
October 2011

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

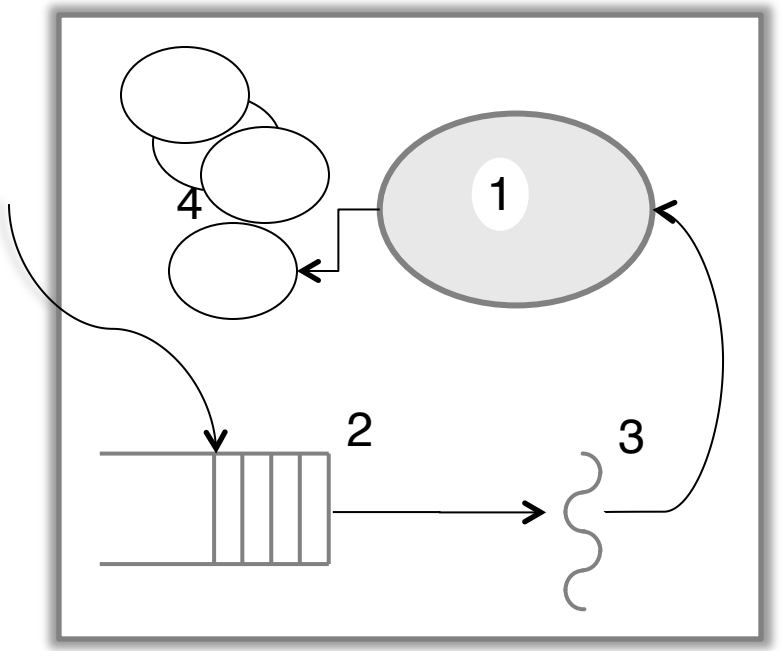
centre de recherche SOPHIA ANTIPOLIS - MÉDITERRANÉE

Outline

1. **Active objects and their limitations**
2. Related Works
3. Proposed solution
4. Experiment
5. Conclusion
6. Current and future work

Active Objects

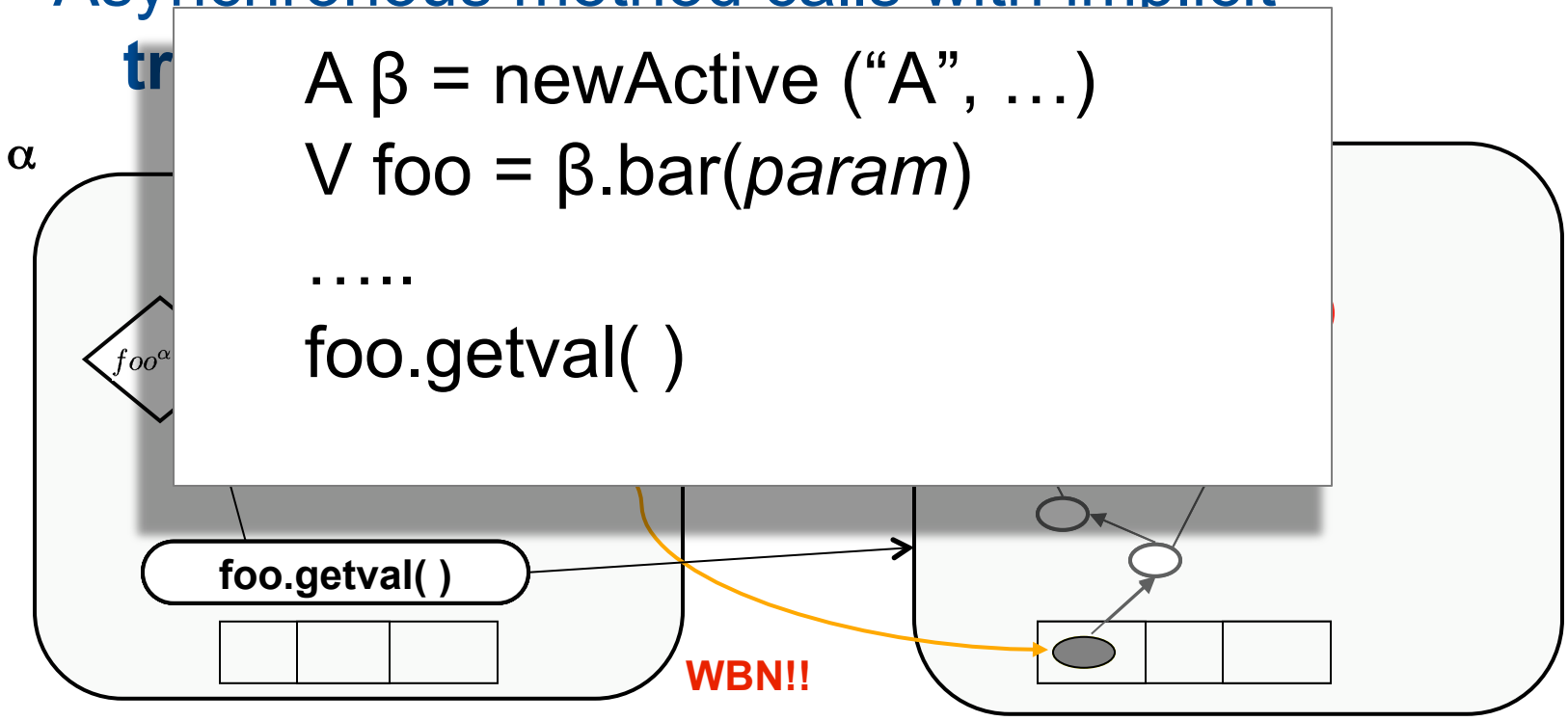
- Asynchronous communication with futures
- Location transparency
- Composition:
  - An active object (1)
  - a request queue (2)
  - one service thread (3)
  - Some passive objects (local state) (4)



ASP /ProActive

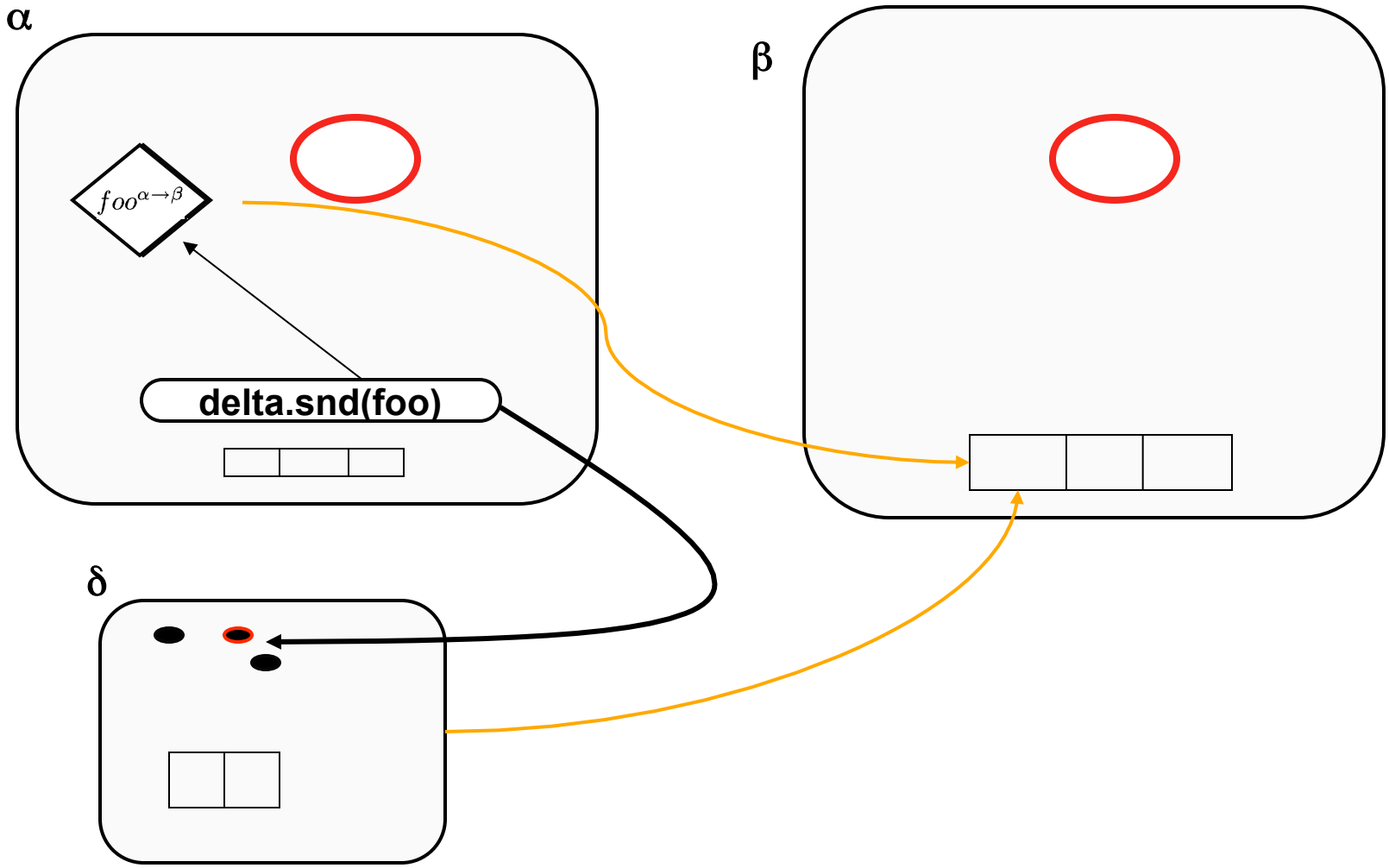
Active objects, asynchronous communication

Asynchronous method calls with implicit

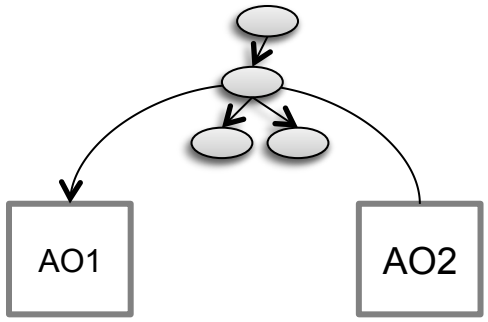


Caromel, D., Henrio, L.: A Theory of Distributed Object. Springer-Verlag (2005)

First Class Futures



Active Objects – Limitations



No data sharing

- Parameters of method calls are passed by value

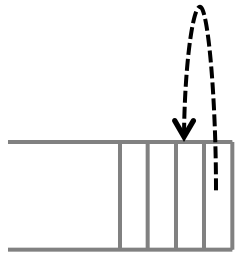
→No data race-condition

→simpler programming + easy distribution

→Slow local parallelism

→Less efficient

No re-entrant calls



- Active object deadlocks by waiting on itself

→Modifications to the application logic

→difficult to program

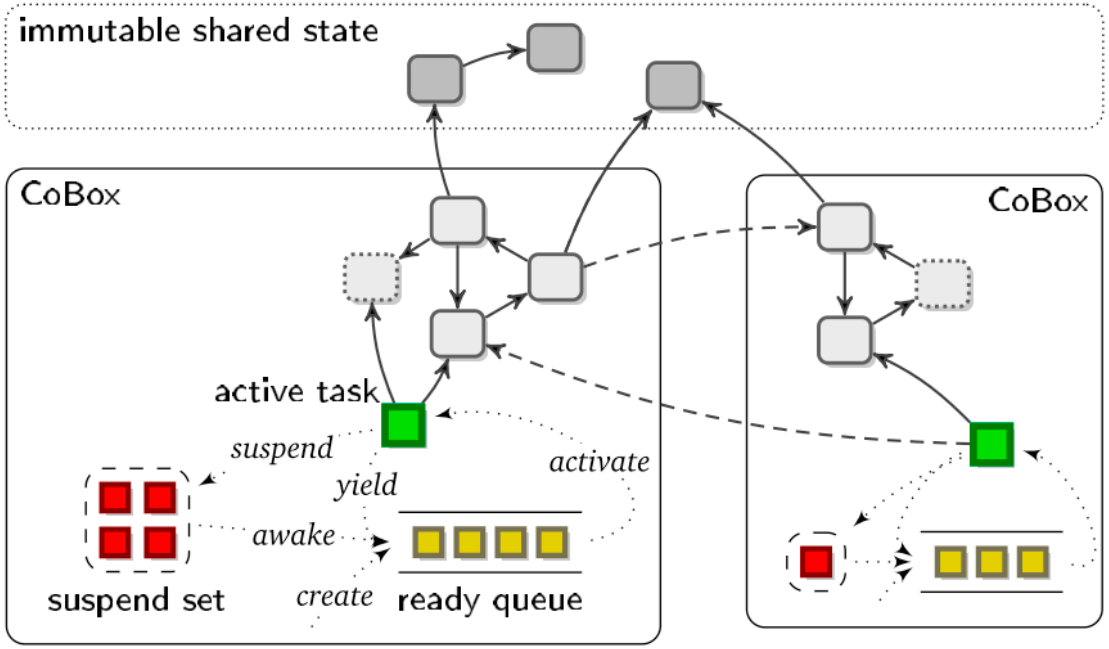
Outline

1. Active objects and their limitations
2. **Related Works**
3. Proposed solution
4. Experiment
5. Conclusion
6. Current and future work

Related Work (I)

Creol an JCobox:

- Active object paradigm & futures



- Cooperative multithreading
  - All requests are served at the same time
  - But only one thread active at a time
  - Explicit release points in the code

➔ Less problem with re-entrance

➔ More difficult to program: less transparency

Related Work (II)

```
/** @compatible append(Object)
 *  @when front != rear */
public Object remove() {
        int x = cell[front];
```

Our "idea": **adapt a simple version of JAC to simple active objects a la ASP to get efficient and easy to program multi-active objects**

- Simulating active objects is possible but not trivial

Outline

1. Active objects and their limitations
2. Related Works
3. **Proposed solution**
4. Experiment
5. Conclusion
6. Current and future work

Proposal: Multi-active object

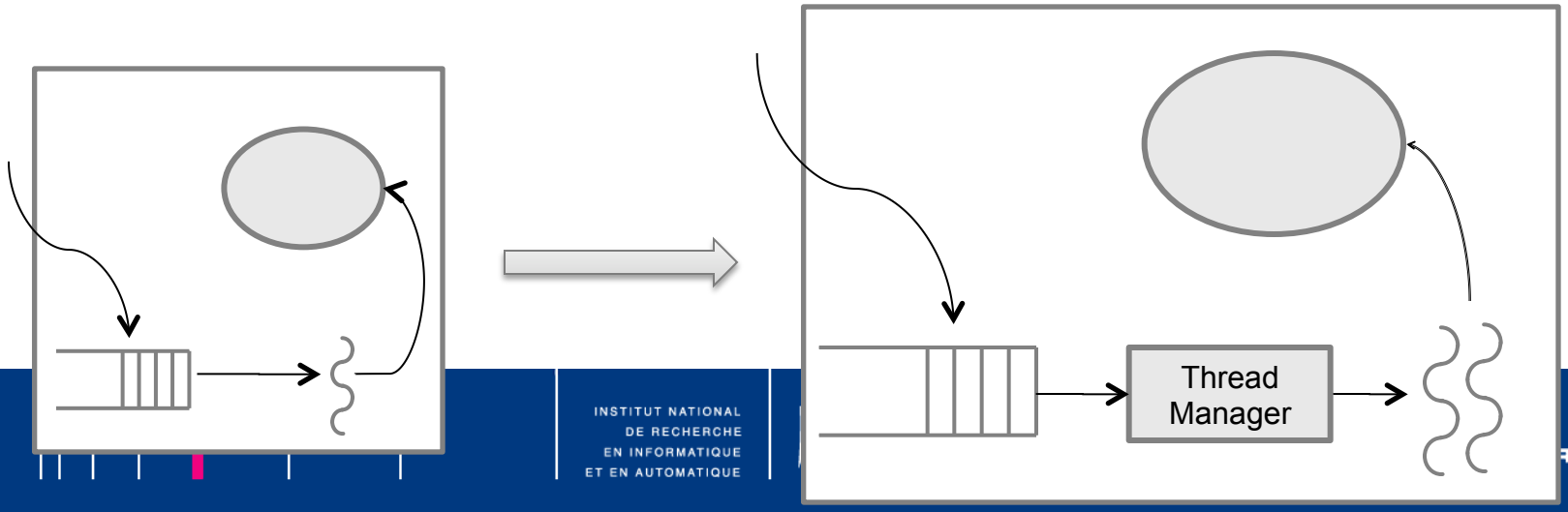
Starting from active objects *à la* ASP

Transparent multi-threading

- Executing compatible requests in parallel

Safe parallel execution

- A thread manager handles the queue



Proposal – Compatibility

Two methods are compatible if either:

a) They do not access the same resources
b) The user “protects” the locations of possible data races

How to express this information?

- Annotate the code

Proposal – Annotations (I)

@Group

- Group identifier
- Self compatible = methods of the group can run in parallel

```
@DefineGroups(
        {
        @Group(name = "GroupFoo", selfCompatible = true),
        @Group(name = "GroupBar", selfCompatible = false)
        }
)
```

Proposal – Annotations (II)

@Compatible

- A list of mutually compatible groups

```
@DefineRules(
        {
        @Compatible({"GroupFoo", "GroupBar"})
        }
)
```

Proposal – Annotations (III)

@MemberOf

- Refers to a group identifier
- Each method belongs to a single group
- Not annotated methods are incompatible with the others

```
@MemberOf("GroupFoo")
public void foo(){
}

@MemberOf("GroupBar")
public void bar(){
}
```

Proposal – Annotation Example

```
@DefineGroups(
        {
        @Group(name = "GroupFoo", selfCompatible = true),
        @Group(name = "GroupBar", selfCompatible = true)
        }
)
```

Groups
(Collection of related methods)

```
@DefineRules(
        {
        @Compatible({"GroupFoo", "GroupBar"})
        }
)
public class SomeClass {

    @MemberOf("GroupFoo")
    public void foo(){
    }

    @MemberOf("GroupBar")
    public void bar(){
    }
```

Rules
(Compatibility relationships between groups)

Memberships
(To which group each method belongs)

Proposal – Thread Manager

Scheduling

- Default: FIFO + compatibility
- Pluggable policies

A policy is a function:

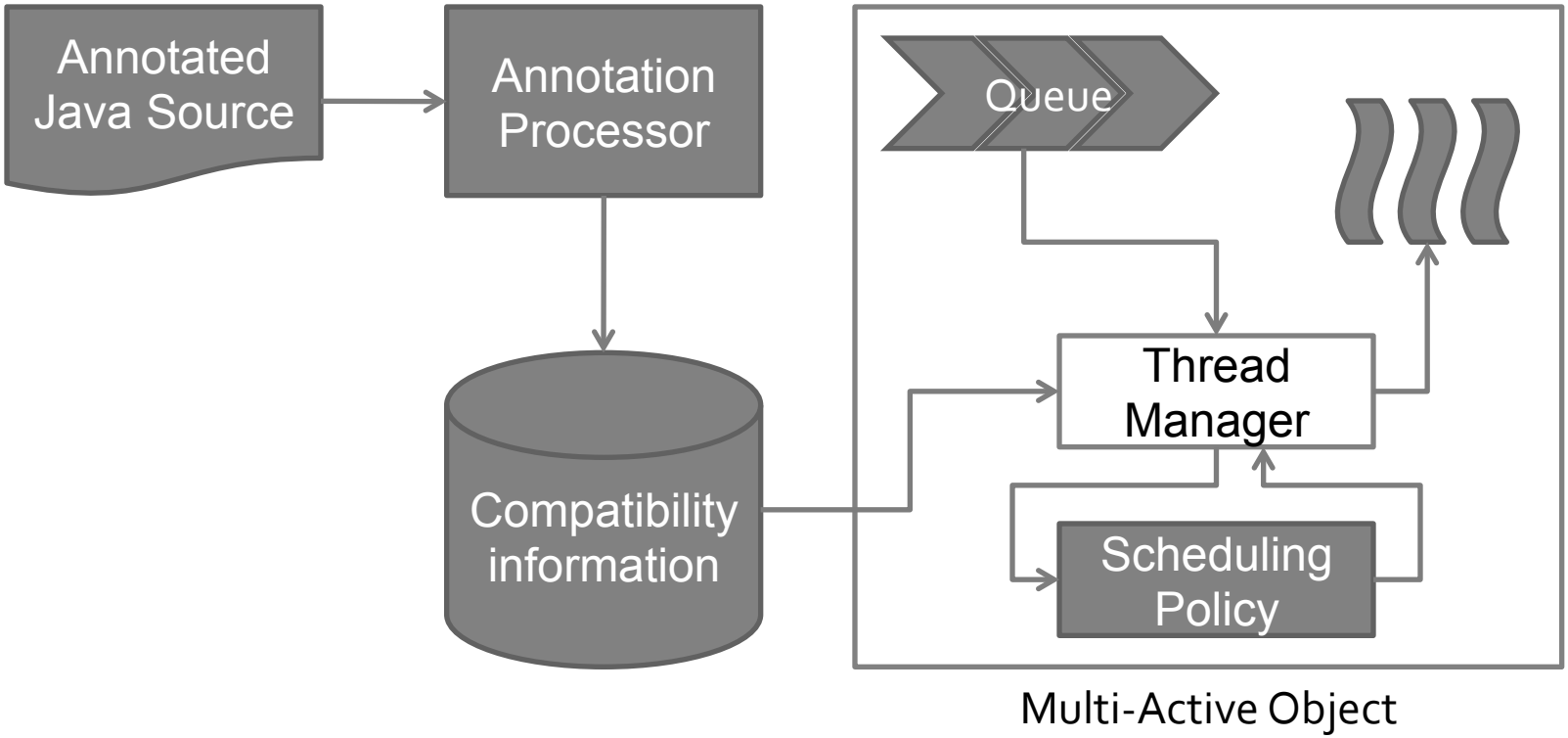
- Input: info from scheduler
- Output: list of requests to be started

Scheduling Policy API provides:

- Static compatibility information
- Scheduler state (request queue, active requests)

```
method runActivity() {
 while (true) {
   if (compatible(requestQueue.peekFirst(),
        activeRequests)) {
     parallelServe(requestQueue.removeFirst());
} } }
```

Proposal – Main Elements



Outline

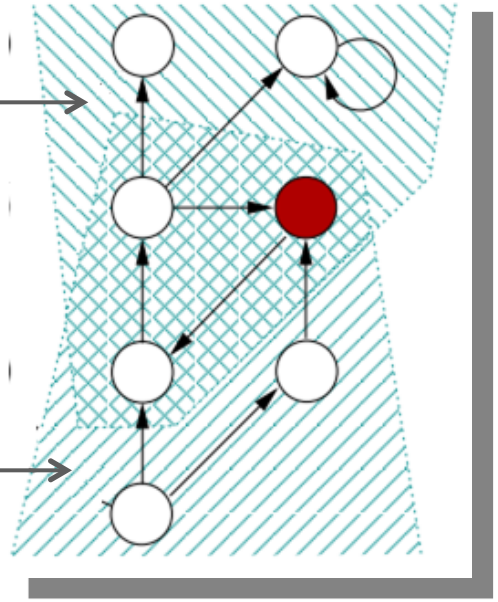
1. Active objects and their limitations
2. Related Works
3. Proposed solution
4. **Experiment**
5. Conclusion
6. Current and future work

Experiment – SCC Search

Strongly connected component search in a distributed graph

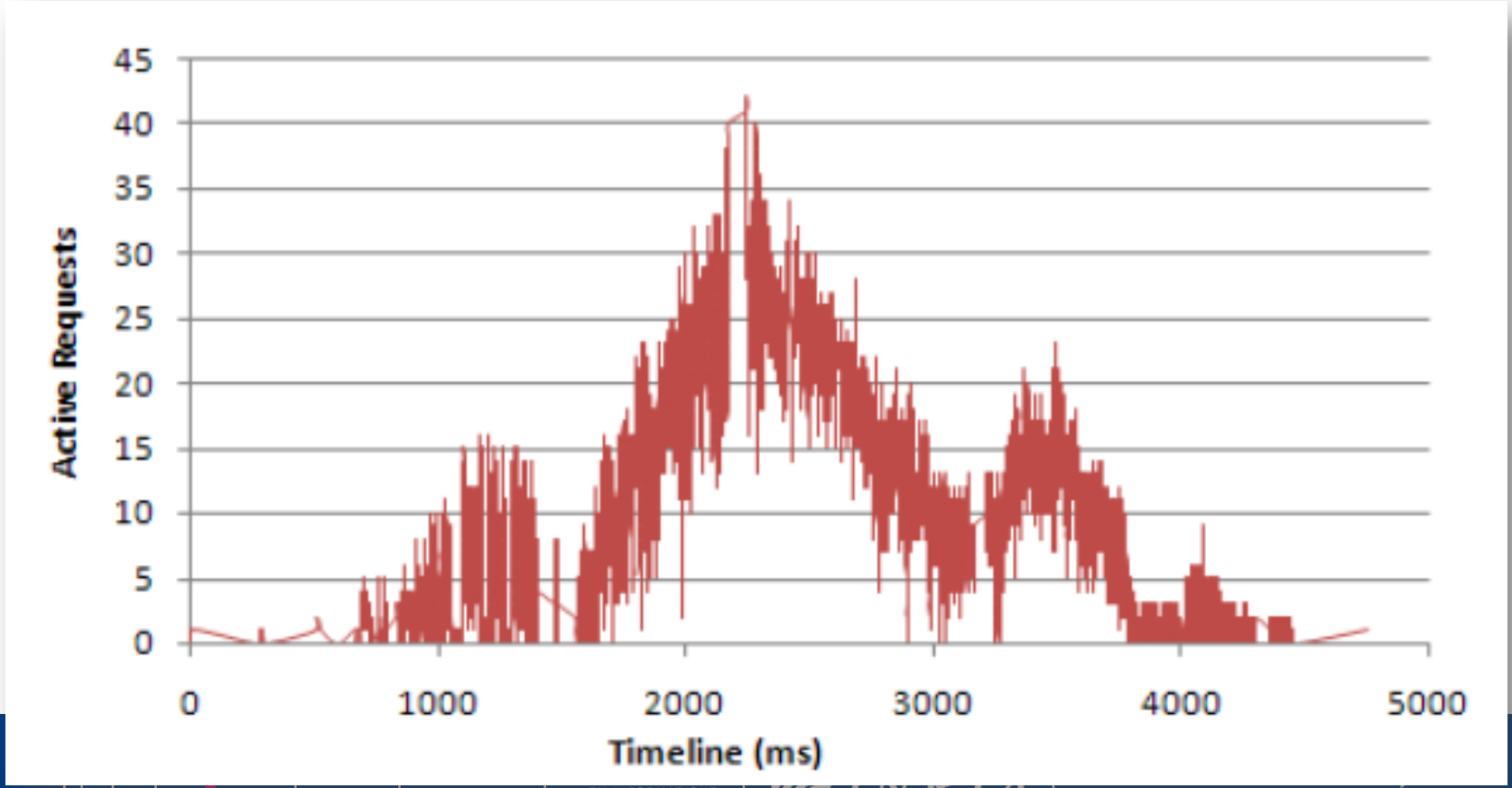
- Divide-and-conquer
- Local and distributed parallelism

```
@DefineGroups(
        {
        @Group(name="ForwardMarking", selfCompatible=true),
        @Group(name="BackwardMarking", selfCompatible=true)
        }
)
@DefineRules(
        {
        @Compatible({"ForwardMarking", "BackwardMarking"})
        }
)
public class GraphWorker implements RunActive { … }
```



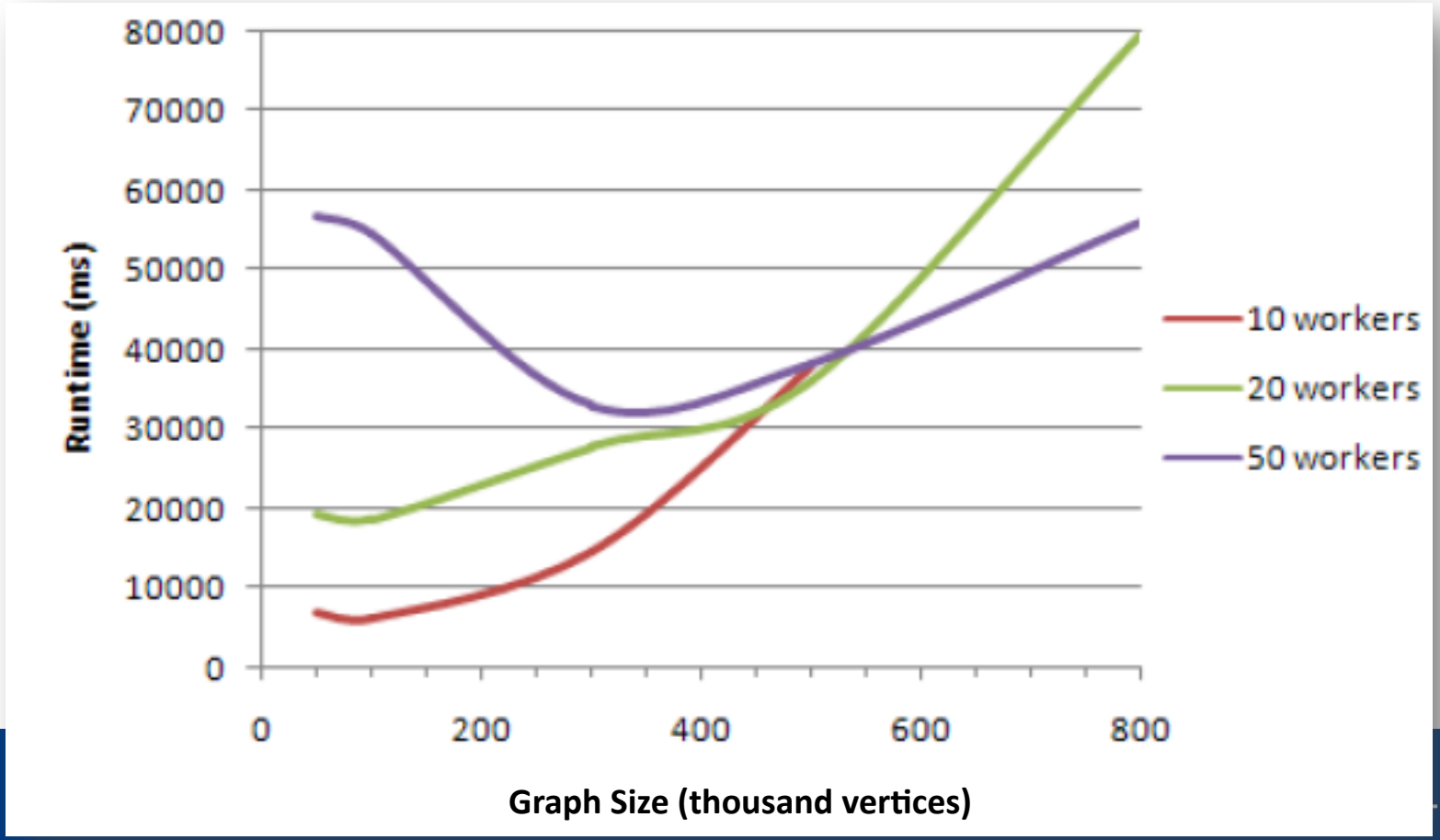
Experiment – Results (I)

Local parallelism



Experiment – Results (II)

Performance



Experiment – Results (III)

Multi-active objects are usable in large-scale applications:

- Easy distributed execution – 80 workers
- Quite large graph sizes – 800,000 vertices

No changes have to be done to the application logic

- Acceptable speedup even without explicit optimization

Outline

1. Active objects and their limitations
2. Related Works
3. Proposed solution
4. Experiment
5. **Conclusion**
6. **Current and future work**

Conclusion

Multi-active objects feature:

- Active object model
  - Easy to program
  - Support for distribution
- Efficient utilization of multi-cores
  - Transparent multi-threading
  - Safe parallel execution
- Possibility to write re-entrant code
- Simple annotations

Current and Future Work (I)

Compatibility

- Create dynamic rules – use parameters
- Common parameter for methods in a group
- Comparison of parameters at runtime

```
@DefineRules({
  @Compatible(value={"data_access", "join"},
        condition="!this.inSplittingZone") })
```

Current and Future Work

Threads

- Too many threads can be harmful
- Limitation of threads – without deadlocks

Formalisation of the multi-active object model

SERVELL

$$\frac{C = \mathcal{R}[Serve(M)] \mapsto f :: [a_i \mapsto f_i] \| C' \qquad ParallelSchedule(M, Futures(C), R) = ([m, f, \iota], R')}{\alpha[F; C; R; \sigma] \;|||\; P \longrightarrow \alpha[F; \iota_0.m(\iota) \mapsto f' \| C; R'; \sigma] \;|||\; P}$$

Questions?

More information:

Adapting Active Objects to Multicore Architectures

Ludovic Henrio, Fabrice Huet, Zsolt István, and Gheorghen Sebestyén International Symposium on Parallel and Distributed Computing *(ISPDC 2011*) - *IEEE*