From Modeling to Deployment of Active Objects - A ProActive backend for ABS

Ludovic Henrio, Justine Rochas

With the contribution of: Fabrice Huet, Zsolt Istvàn

Agenda

I. Active Object Programming models

II. Multi-active Objects: Principles

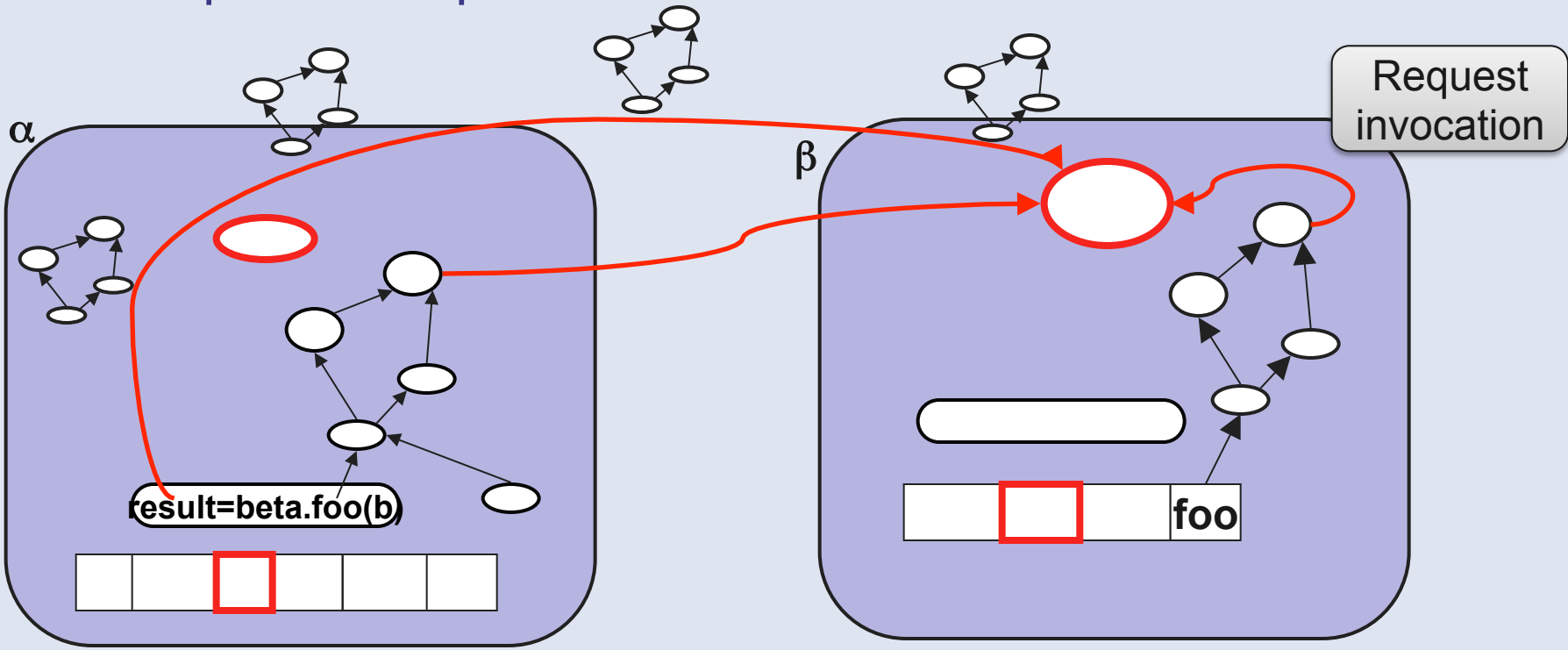
III. Scheduling in Multi-active Objects

IV. A ProActive backend for ABS

V. Conclusion and Future Works

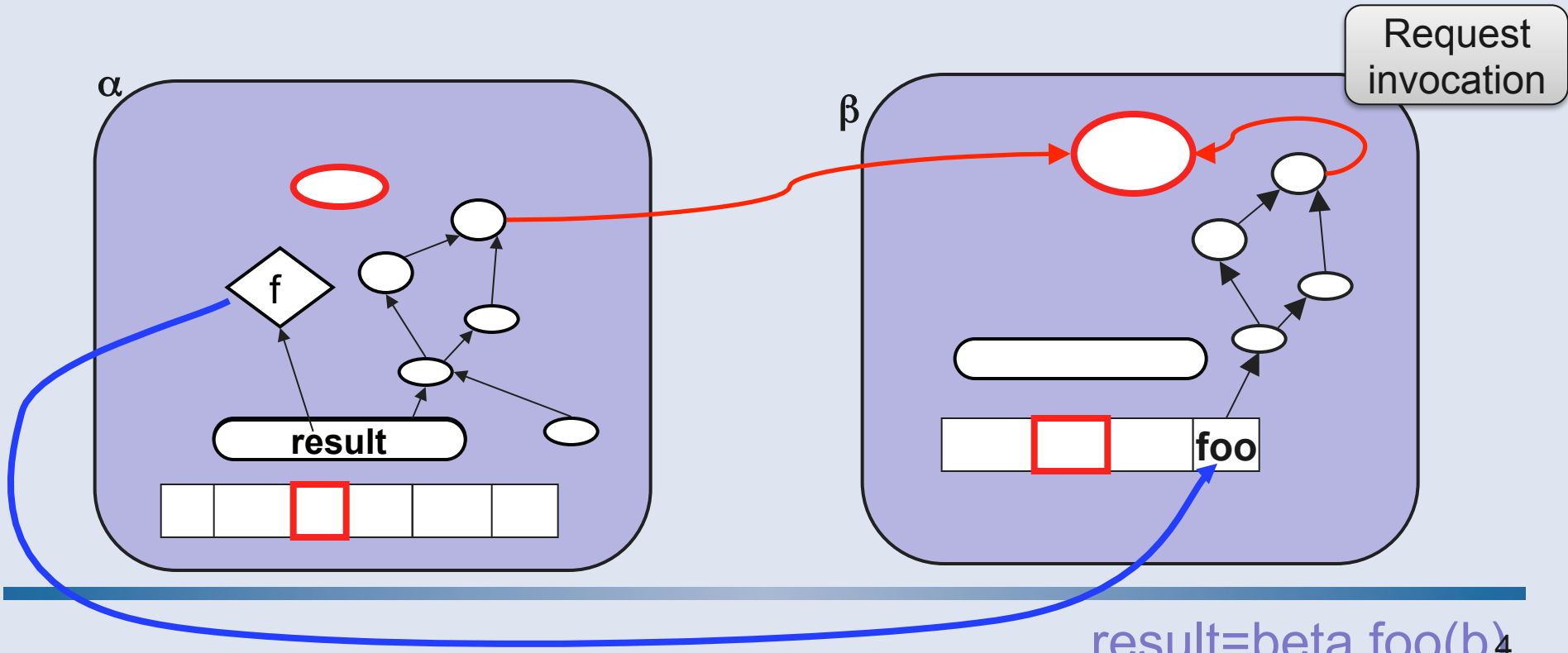
ASP/ProActive

- No race condition: each object manipulated by a single thread
- Active and Passive objects
- Asynchronous method calls ; request queue
- With implicit transparent futures



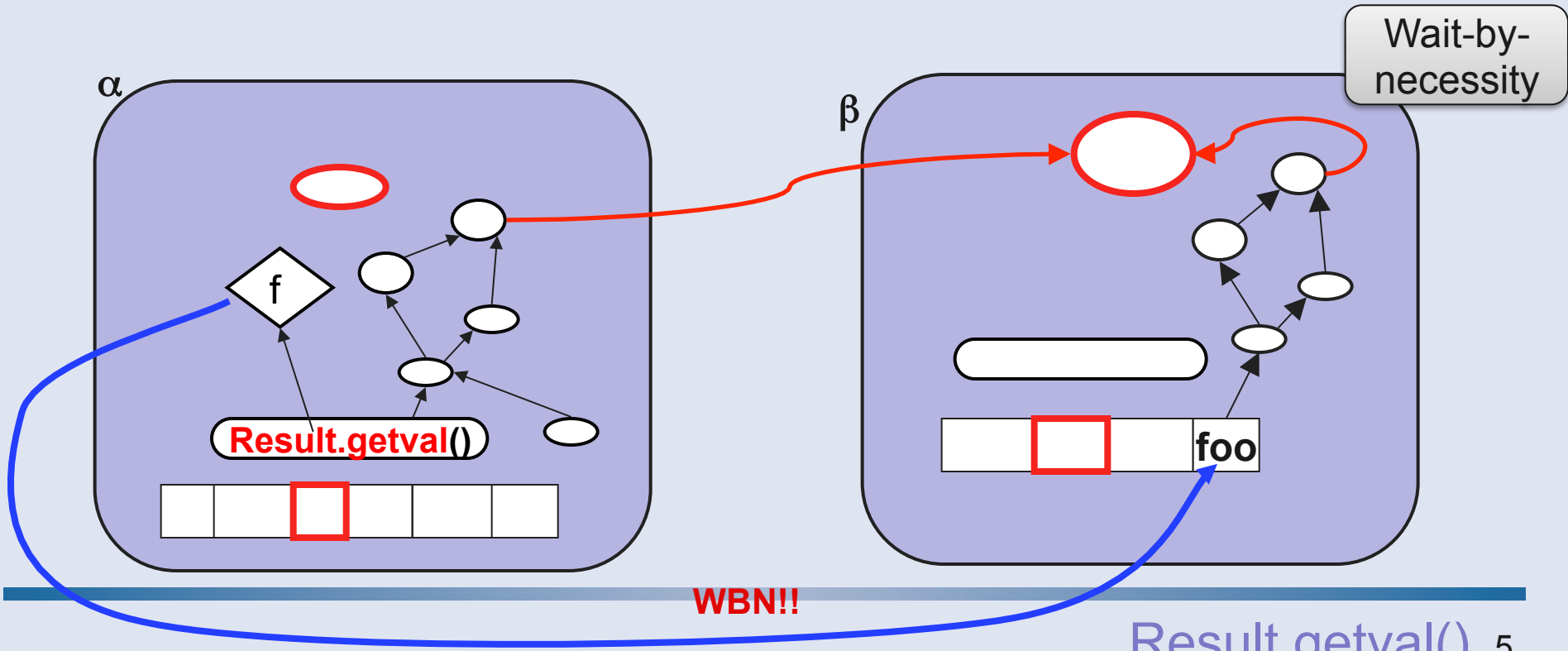
ASP/ProActive

- No race condition: each object manipulated by a single thread
- Active and Passive objects
- Asynchronous method calls ; request queue
- With implicit transparent futures



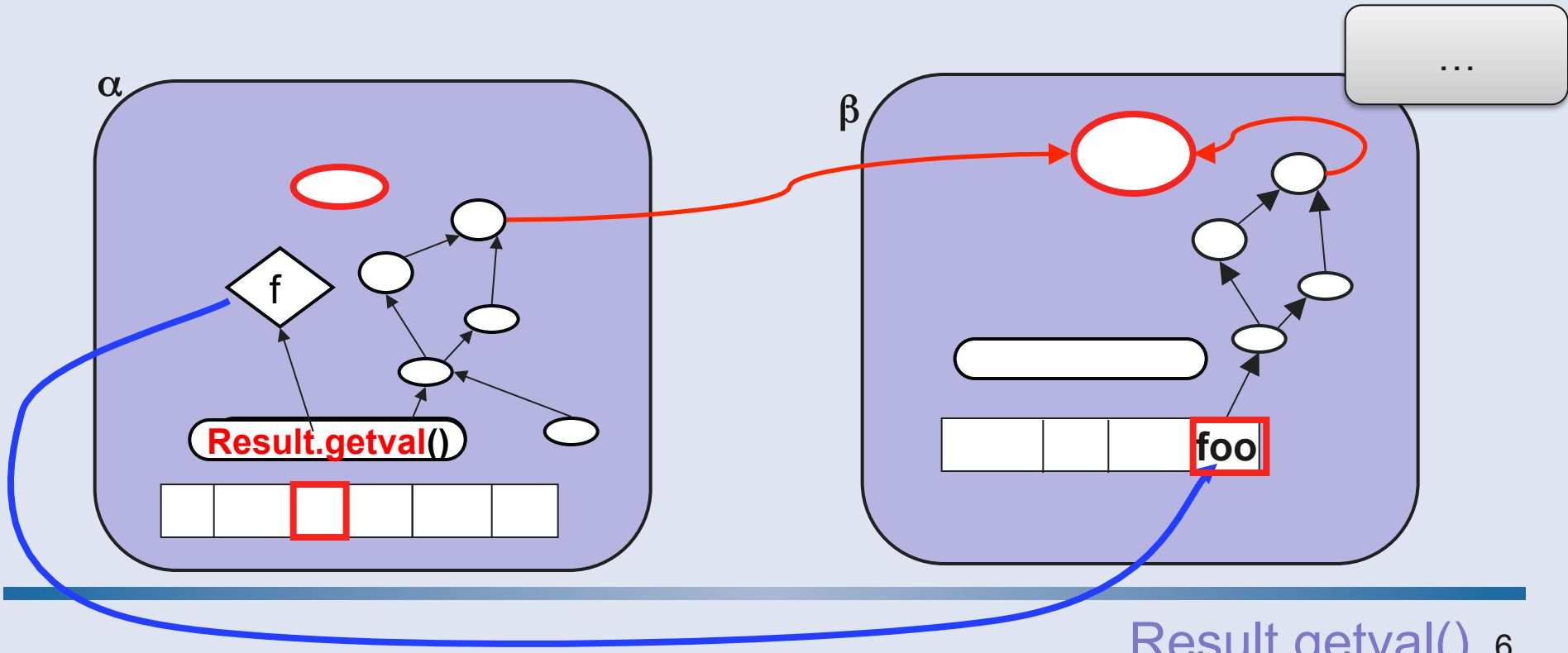
ASP/ProActive

- No race condition: each object manipulated by a single thread
- Active and Passive objects
- Asynchronous method calls ; request queue
- With implicit transparent futures



ASP/ProActive

- No race condition: each object manipulated by a single thread
- Active and Passive objects
- Asynchronous method calls ; request queue
- With implicit transparent futures



ASP/ProActive

- No race condition: each object manipulated by a single thread
- Active and Passive objects
- [illegible]
- [illegible]

A beta = newActive (“A” ,);

ProActive is a Java library

ASP is a “calculus”

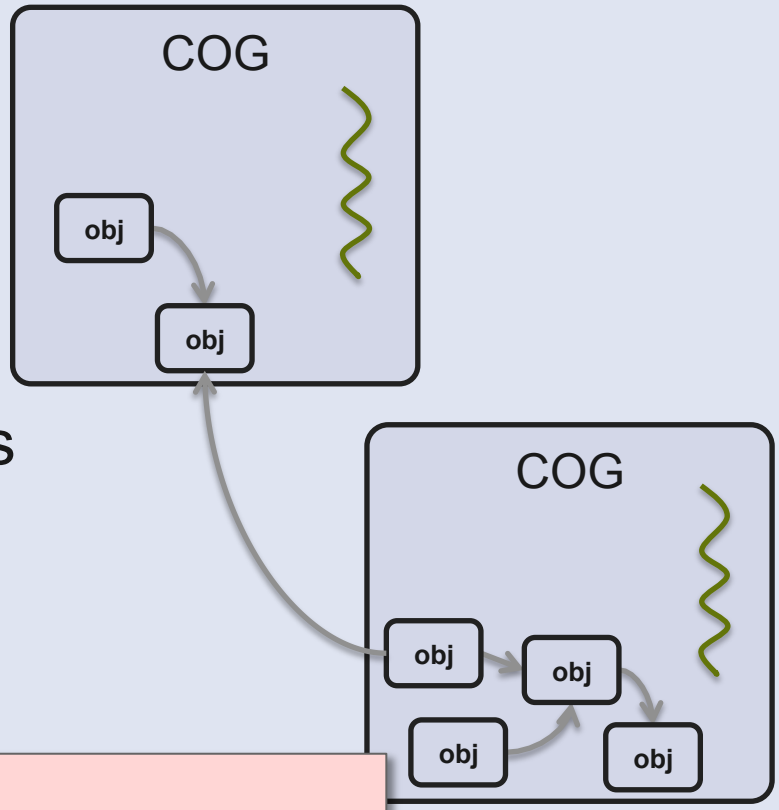
Active objects are the unit of *distribution* and *concurrency (one thread per AO / no data shared)*

Futures are transmitted between AOs and accessed transparently

Result.getval()

ABS

- COGs (set of objects)
- Each object is active (can be invoked remotely)
- Cooperative scheduling
- Explicit syntax for asynchronous call and future access




```
A a = new cog A();
B b = new cog B();
b!foo(a);b!foo(a);
```

```
foo(A a) {
  Fut<V> vFut = a!bar(p);
  await vFut?;
  V v = vFut.get;
}
```

```
foo(A a) {
  Fut<V> vFut = a!bar(p);
  await vFut?;
  V v = vFut.get;
}
```

ABS versus ProActive

Creol, ABS, JCobox

- Explicit asynchronous calls

```
object.method() // synchronous
object!method() // asynchronous
```

- Explicit futures

```
Fut<T> future = object!method();
T t = future.get; // blocks
```

ASP/ProActive

- Transparent asynchonous calls

```
object.method() // synchronous
                or asynchronous
```

- Transparent first class futures

```
T future = object.method();
```

Single-threaded

- ASP/ProActive

Cooperative

- Creol
- JCoBox
- ABS

Multi-threaded

- Multi-active Objects: MultiASP/new ProActive

Local Concurrency

Agenda

I. Active Object Programming models

II. Multi-active Objects: Principles

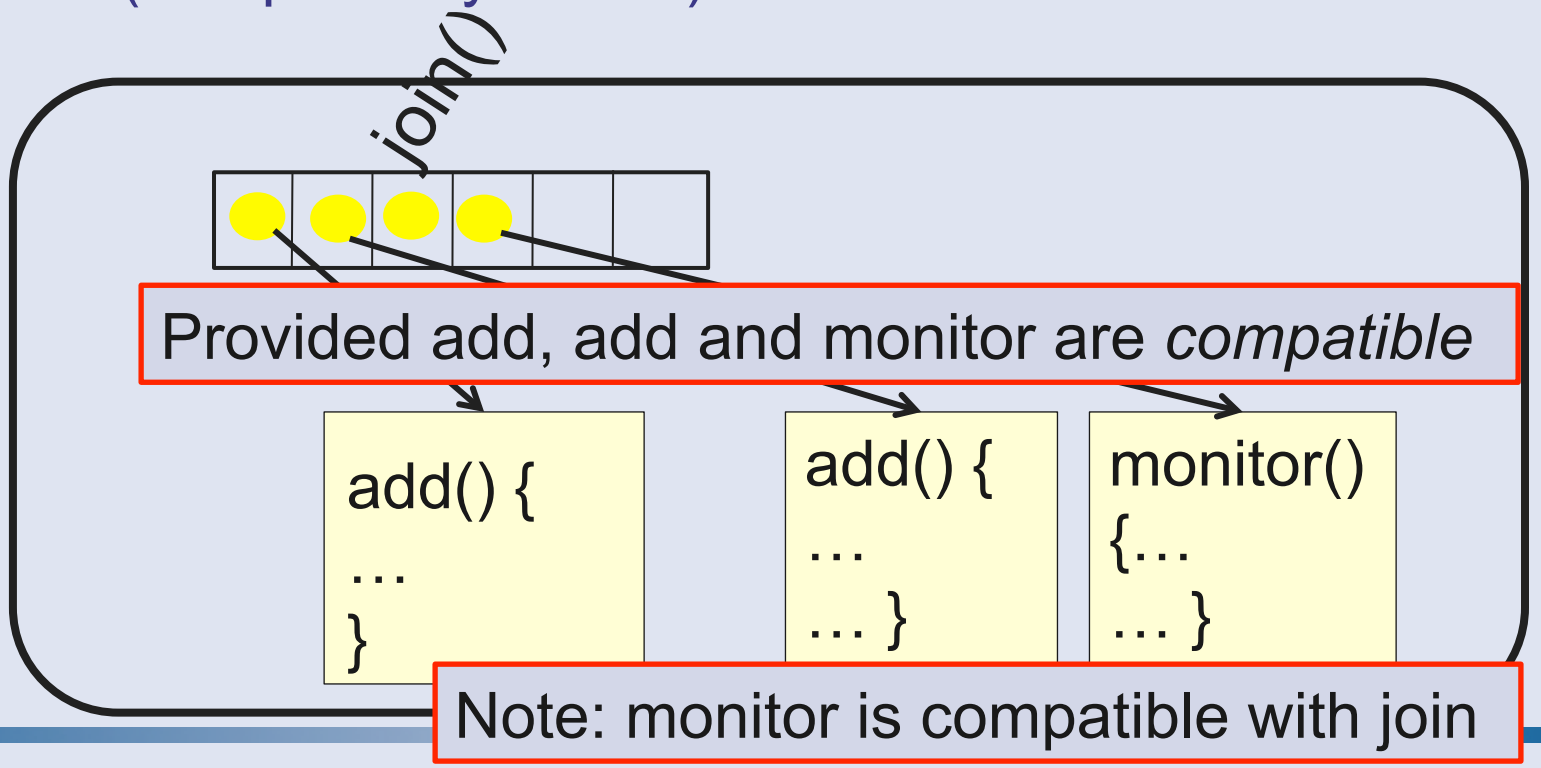
III. Scheduling in Multi-active Objects

IV. A ProActive backend for ABS

V. Conclusion and Future Works

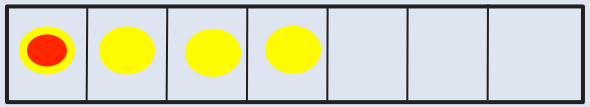
Multi-active objects

- A programming model that mixes local parallelism and distribution with high-level programming constructs
- Execute several requests in parallel but in a *controlled manner* (compatibility notion)



Scheduling Requests

- An « optimal » request policy that « maximizes parallelism »:
  - ➜ Schedule a new request as soon as possible (when it is compatible with all the served ones)
  - ➜ Serve it in parallel with the others
  - ➜ Serves
    - Either the first request



Compatibility =

requests can execute at the same time

and can be re-ordered

Declarative concurrency by annotating methods

```
@DefineGroups({
  @Group(name="join", selfCompatible=false)
  @Group(name="routing", selfCompatible=true)
})
```

Groups
(Collection of related methods)

```
@DefineRules({
  @Compatible({"join", "monitoring"})
  @Compatible({"routing", "monitoring"})
})
```

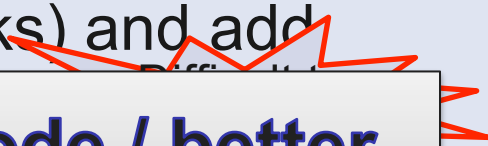
Rules
(Compatibility relationships between groups)

```
public class Peer {
...
@MemberOf("join")
  public JoinResponse join(Peer other) { ... }

@MemberOf("routing")
  public void add(Key k, Serializable value) { ... }

@MemberOf("routing")
  public Serializable lookup(Key k) { ... }
}
```

Memberships
(To which group each method belongs)

Hypotheses and programming methodology

- **We trust the programmer**: annotations supposed correct
  static analysis or dynamic checks should be applied in the future
- Without annotations, a multi-active object runs like an active object
- If more parallelism is required:
  1. Add annotations for non-conflicting methods
  2. Declare dynamic compatibility (depending on runtime values, eg request parameters)
  3. Protect some memory access (e.g. by locks) and add



More parallelism → More complex code / better performance

Agenda

I. Active Object Programming models

II. Multi-active Objects: Principles

III. Scheduling in Multi-active Objects

IV. A ProActive backend for ABS

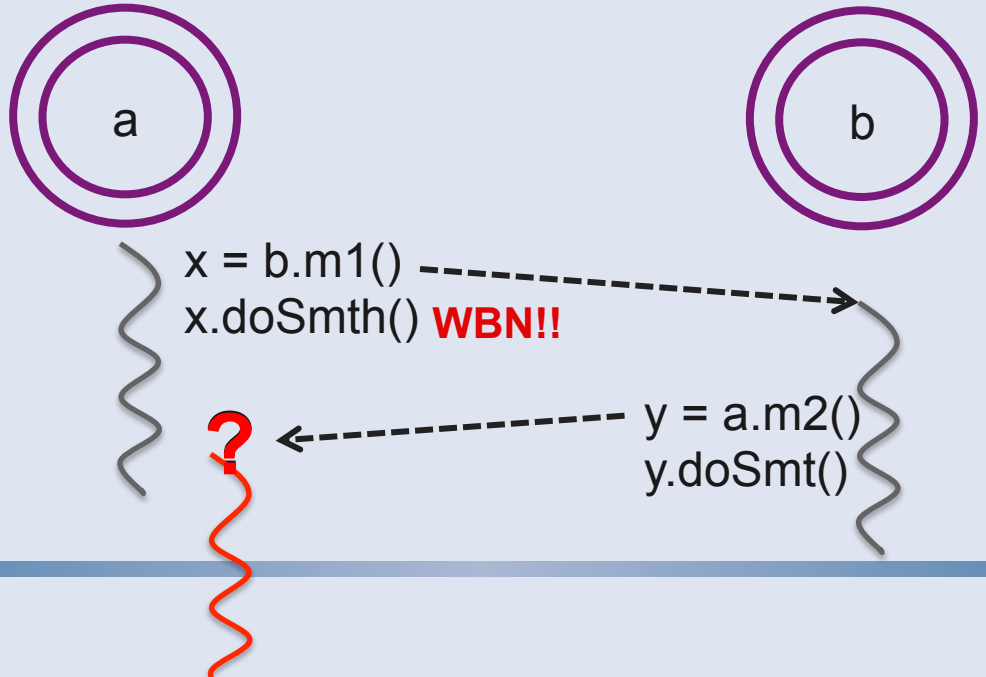
V. Conclusion and Future Works

Thread Limitation per Multi-active Object

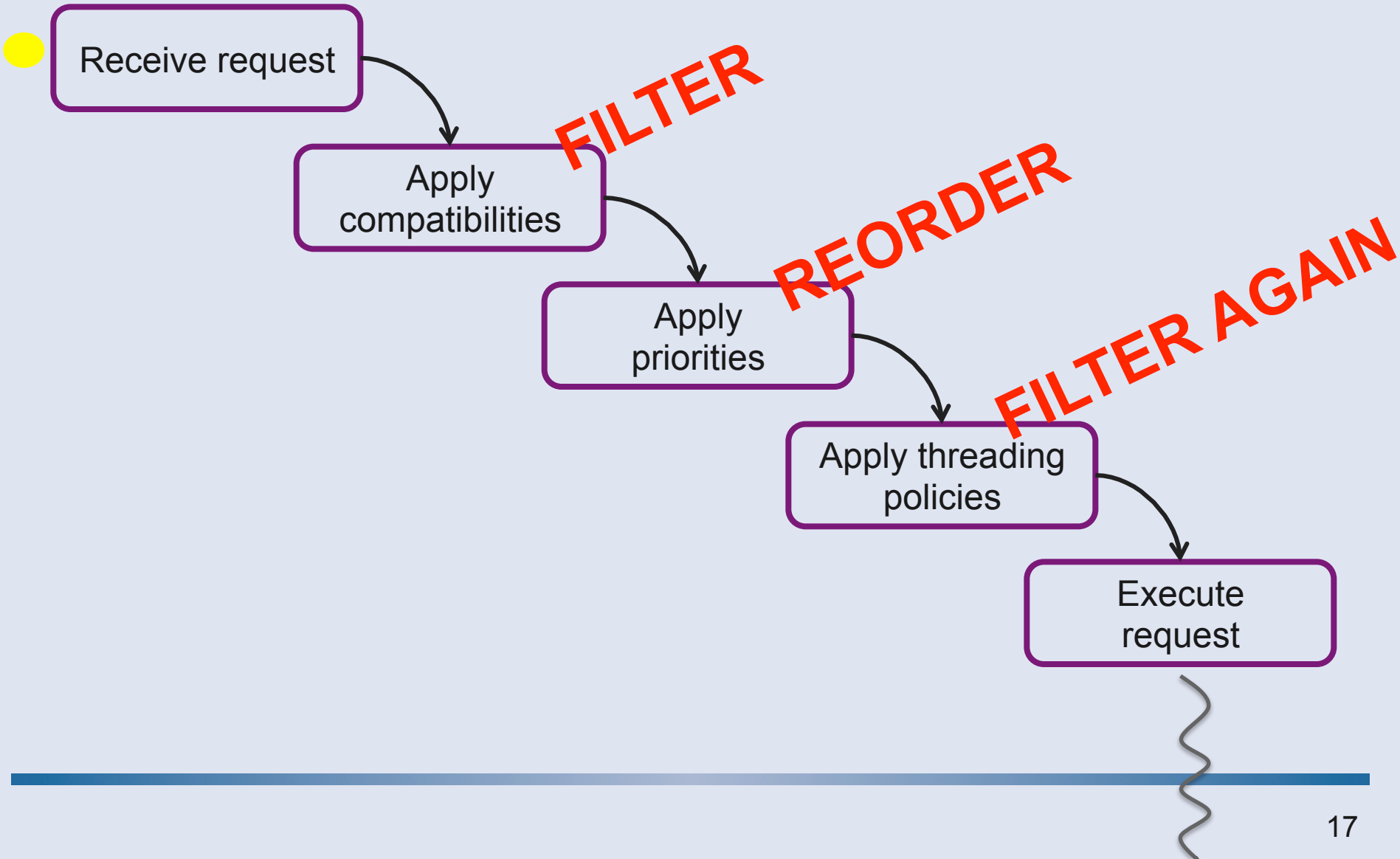
- Too many threads harm:
  - memory consumption,
  - too much concurrency wrt number of cores

```
@DefineThreadConfig(threadPoolSize=1, hardLimit=true)
public class MyObject {
    …
}
```

false

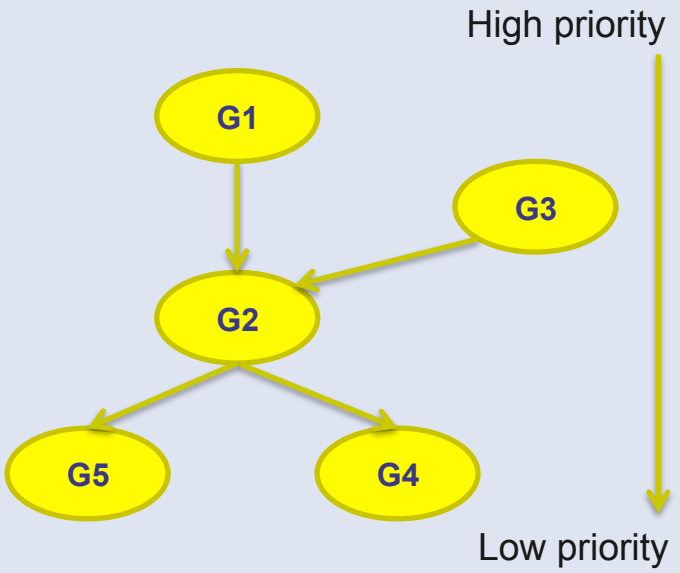


Multi-active Object Scheduling - Workflow



Priority Specification Mechanism

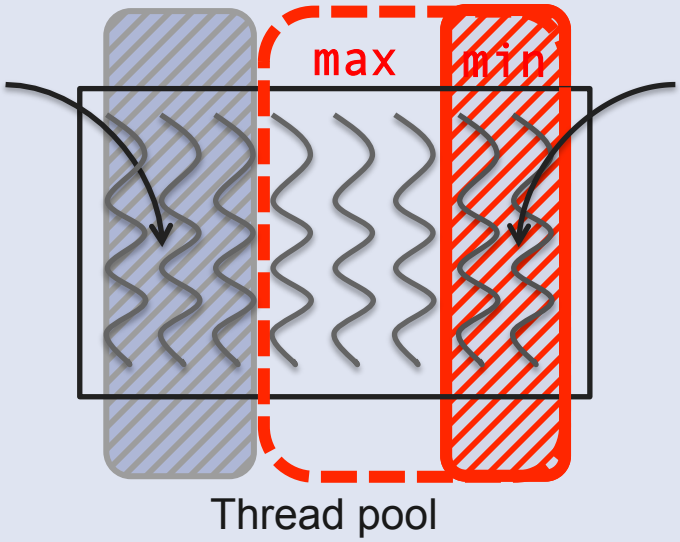
```
@Group(name="G1", selfCompatible=true)
    …
@PriorityOrder({
     @Set(groupNames = {"G1"}),
     @Set(groupNames = {"G2"}),
     @Set(groupNames = {"G5","G4"})
}),
@PriorityOrder({
     @Set(groupNames = {"G3"}),
     @Set(groupNames = {"G2"})
})
public class MyObject {
    …
}
```



Thread Limitation per Group

```
@Group(name=" routing ", minThreads=2, maxThreads=5)
public class MyObject {
    …
}
```

Threads never used by the **routing group**



Threads never used by **other groups**

Summary of the MAO programming model

Key notions

- Groups of requests
- Compatibility
  - possibly decided dynamically
  - between groups
  - between requests from the same group

Key features

- Global thread limit (soft or hard)
- Upper and lower bounds per group
- Priorities among compatible requests

Agenda

I. Active Object Programming models

II. Multi-active Objects: Principles

III. Scheduling in Multi-active Objects

IV. A ProActive backend for ABS

V. Conclusion and Future Works

Motivation

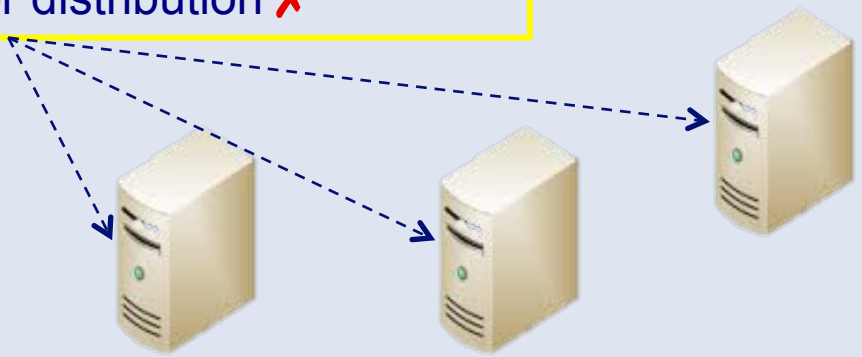
ProActive – Multi-active Objects

Development and deployment of distributed applications ✔

ABS – Abstract Behavioral Spec. Language

Modeling of distributed applications ✔
Verification tools ✔
Java Translator ✔
No support for distribution ✗

OBJECTIVE
Provide distributed deployment to ABS using ProActive



From ABS to ProActive: Challenges

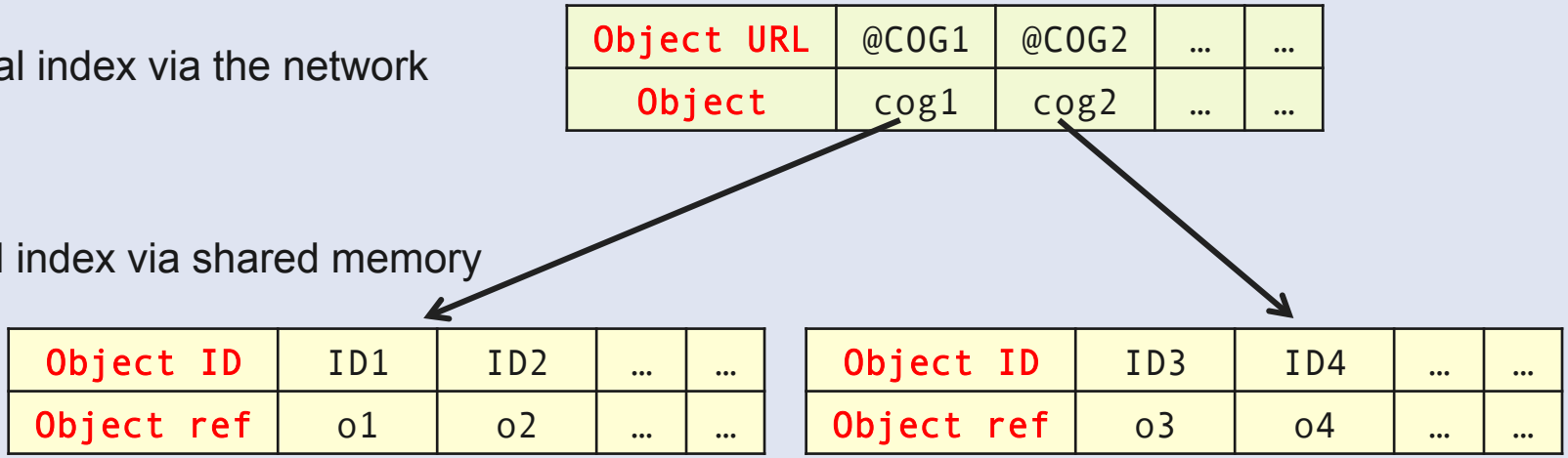
	ABS	MultiASP/ProActive
Active object model	Object group	Non uniform
Asynchronous method calls and futures model	Explicit	Transparent
Threading model	Cooperative	Multi-threaded

Towards translation of ABS in ProActive

- Select active objects
  - ABS object group (COG) = ProActive active object
  - Entry point to the local memory space
- Hierarchical indexing of objects

① Global index via the network

② Local index via shared memory



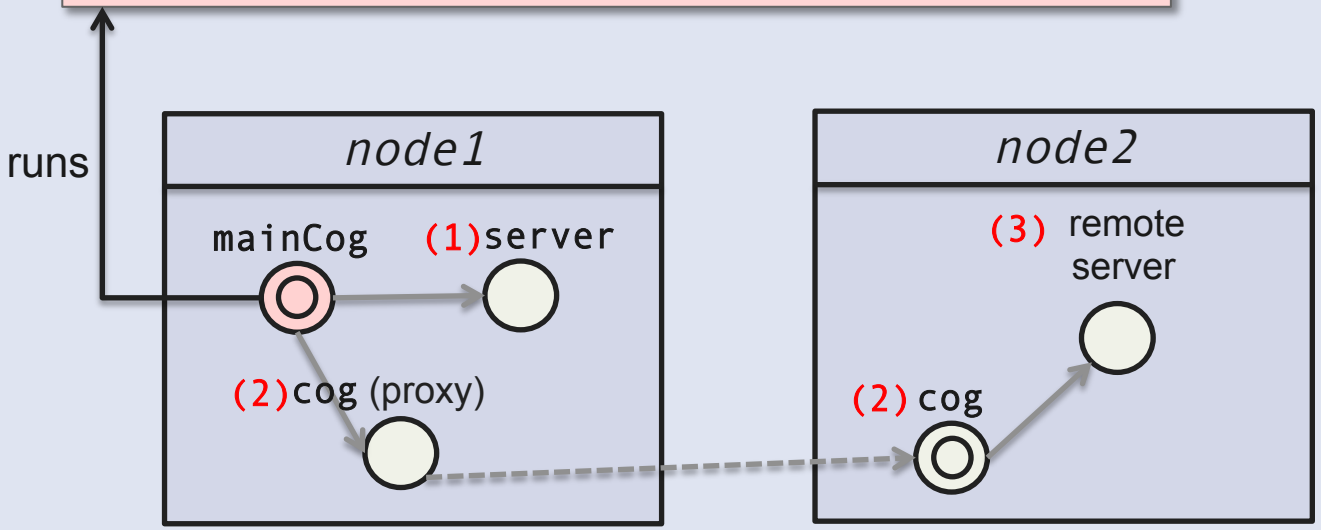
Translation of a *new cog* statement

ABS code:

```
Server server = new cog Server()
```

Translation in ProActive:

```
Server server = new Server()                    (1)
COG cog = newActive(COG.class, {}, node2)       (2)
cog.registerObject(server)                      (3)
```



object active object local reference remote reference

From ABS to ProActive: Challenges

	ABS	MultiASP/ProActive
Active object model	Object group	Non uniform
Asynchronous method calls and futures model	Explicit	Transparent
Threading model	Cooperative	Multi-threaded

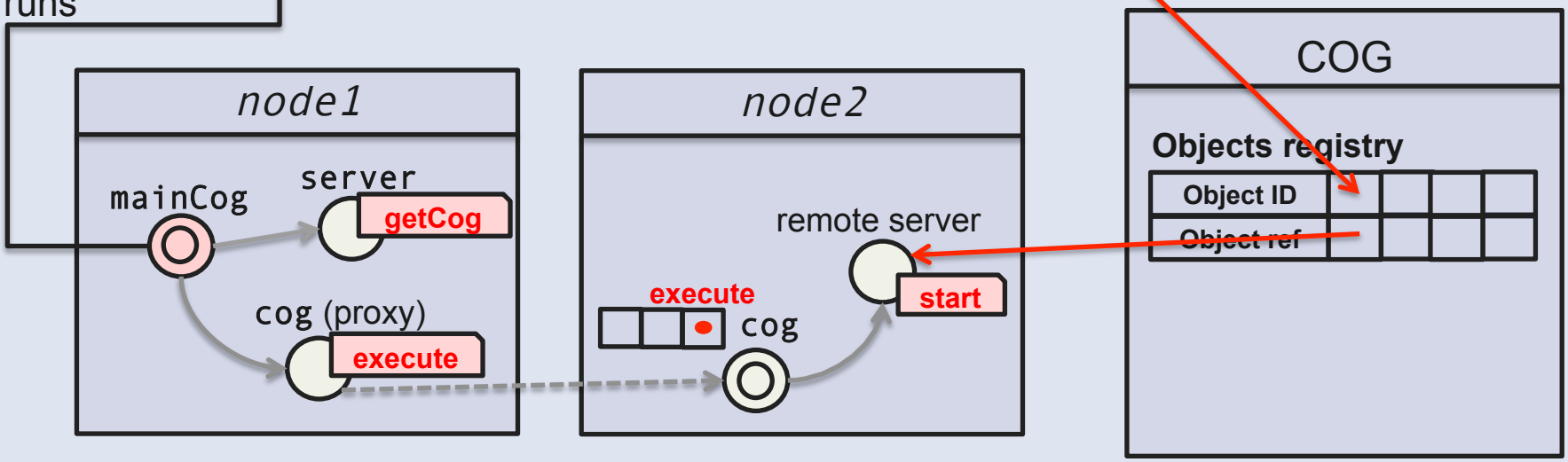
Translation of an Asynchronous Method Call

ABS code:

```
server!start()
```

Translation in ProActive:

```
server.getCog().execute("start", {}, server.getID())
```



object | active object | local reference | remote reference

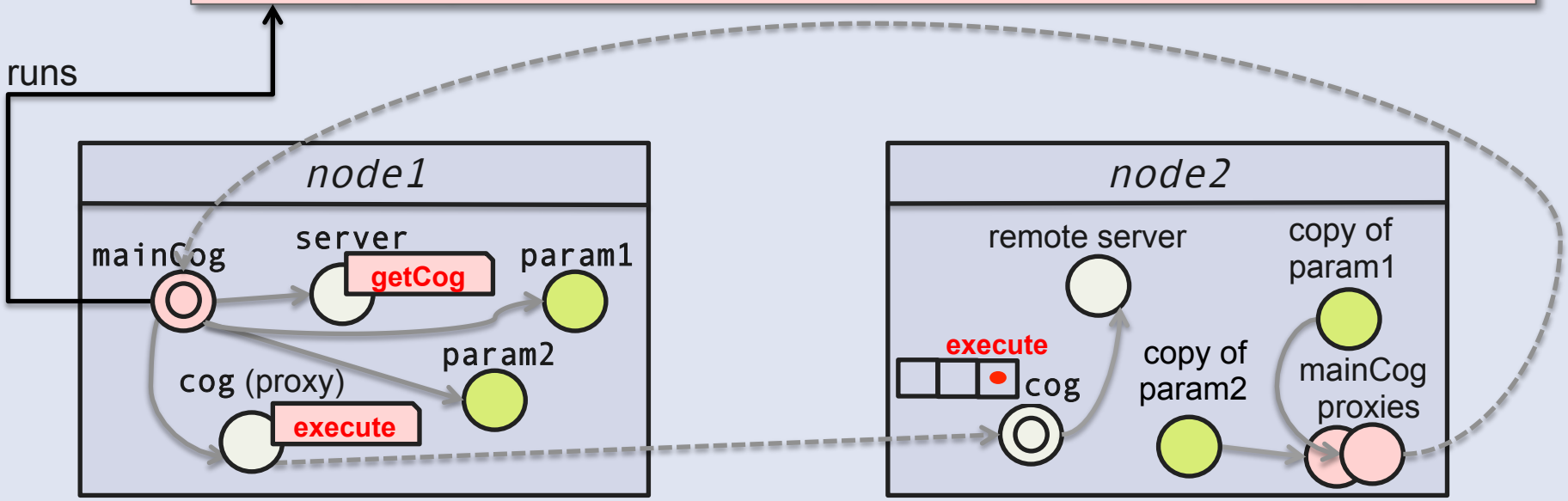
Asynchronous Method Call with Parameters

ABS code:

```
server!start(param1, param2)
```

Translation in ProActive:

```
server.getCog().execute("start", {param1, param2}, server1.getID())
```



object | active object | local reference | remote reference

From ABS to ProActive: Challenges

	ABS	MultiASP/ProActive
Active object model	Object group	Non uniform
Asynchronous method calls and futures model	Explicit	Transparent
Threading model	Cooperative	Multi-threaded

Translation of an *await* statement

ABS code:

```
Fut<Bool> readyFut = server!start()          (1)
await readyFut?                              (2)
```

Translation in ProActive:

```
PAFuture.getFutureValue(readyFut)            (2)
```

```
@DefineGroups({
  @Group(name="scheduling", selfCompatible=true)
})
@DefineThreadConfig(threadPoolSize=1, hardLimit=false)
public class COG {
  @MemberOf("scheduling")
  public ABSType execute(…) {
  }
}
```

Translation of a *get* statement

ABS code:

```
Fut<Bool> readyFut = server!run()          (1)
Bool ready = readyFut.get                  (2)
```

Translation in ProActive

```
this.getCOG().switchHardLimit(true);                (2)
Boolean ready = PAFuture.getFutureValue(readyFut);
this.getCOG().switchHardLimit(false);
```

Change temporarily

The default value

```
@DefineThreadConfig(threadPoolSize=1, hardLimit=false)
public class COG {
    …
}
```

Direct Modifications for Distribution

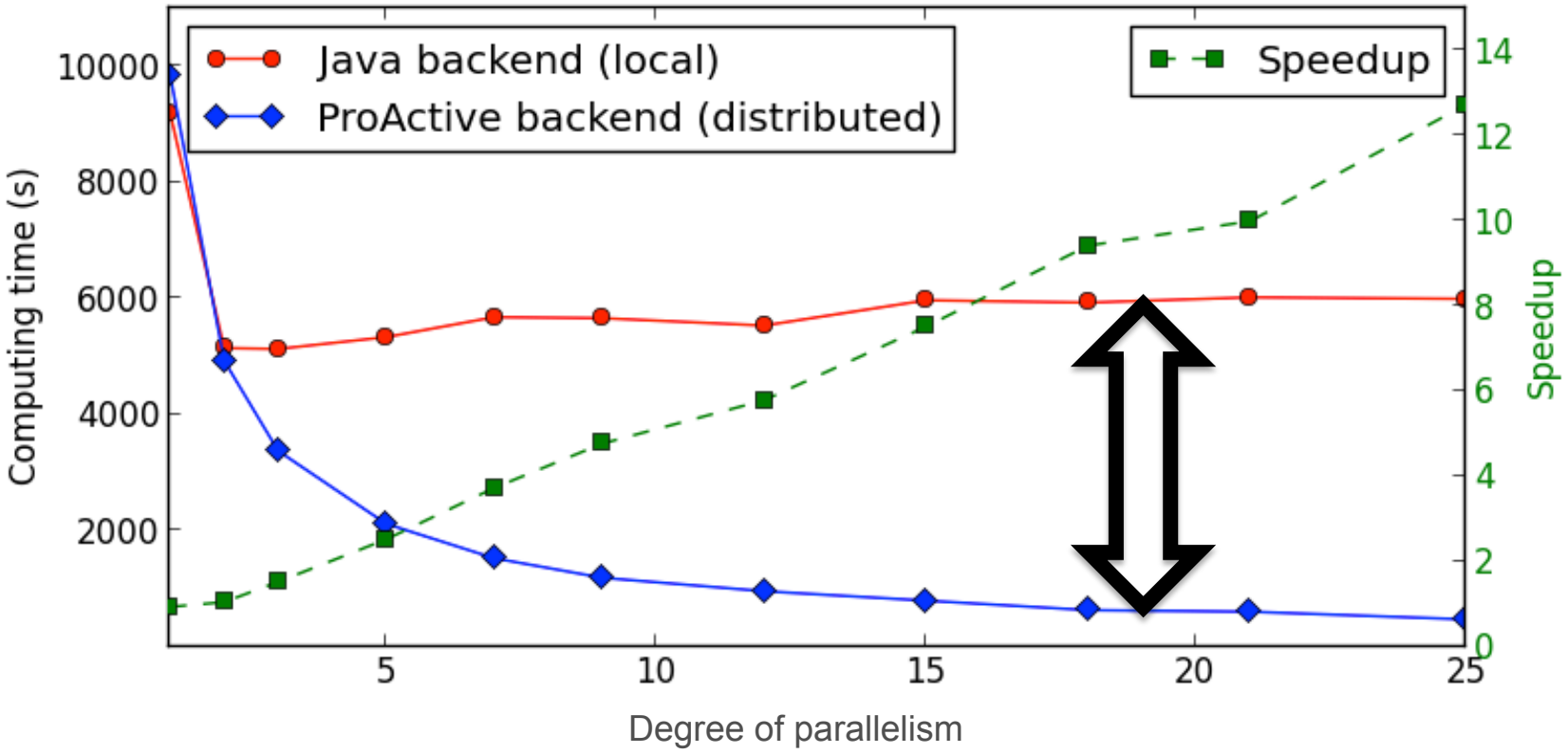
- Serialization
  - Most classes implement now "***Serializable***"
  - Optimization: "***transient***" fields

- Deployment
  - **Node specification** added in the ABS language

ABS code:

```
Server server = new cog "slaves" Server();
```

Experimental evaluation: DNA matching algorithm



The ProActive backend: A Fully Working Tool

- Automated compilation & deployment of ABS programs
- Significant speedup from local programs
- Overhead < 10% from a native ProActive application

Formalization: Correctness of translation

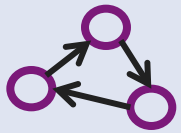
- Operational semantics of MultiASP/ProActive
- Translational semantics: ABS ➔ MultiASP/ProActive

- Proven:
  - Translation simulates any ABS execution ✔
  - Translation corresponds to a possible ABS execution

- Restrictions:
  - FIFO request service
  - Causally ordered communications

Conclusion

- Multi-active objects implemented on top of ProActive
  - A programming model for local concurrent and global distributed objects
- Many case studies & benchmarks
  - NAS, Content Addressable Network, GCM components, ProActive backend for ABS
- Visual *post-mortem* debugger of Multi-active Object app.
- Formal proofs (based on semantics)
  - « maximal parallelism »
  - Correctness of ABS translation
- Next steps:
  - Prove stronger properties, ongoing formalisation in Isabelle/HOL
  - Find deadlocks using behavioural types (ongoing PhD)
  - Design a recovery protocol for MAO



Questions?

Related publications:

1. Multi-threaded Active Objects. Ludovic Henrio, Fabrice Huet, and Zsolt István - In COORDINATION 2013.
2. Declarative Scheduling for Active Objects paper. Ludovic Henrio and Justine Rochas - SAC 2014.
3. Justine Rochas, Ludovic Henrio. A ProActive Backend for ABS: from Modelling to Deployment. Inria Research Report 2014.