Stopping Safely Hierarchical Distributed Components: Application to GCM

Ludovic Henrio and Marcela Rivera

July 2008

Oasis Project, INRIA - Sophia Antipolis

Agenda

- **Introduction and objectives**
  - Assumptions
  - Characteristics of the algorithm
- Description of algorithm
  - Example
- Conclusion
  - Extension and futures works

Introduction

- When is a system reconfigurable?
- How to stop a hierarchical components assembly?
  - An algorithm for reaching a *safe state*.
- First phase of the reconfiguration of components.

Objectives

Stopping a distributed component system.

- Without incoherencies in the system.
- Each inner components is in a *safe state* after the process.
- Stopping a system without livelock and deadlock.
- Don’t block the system if it can’t be stopped safely.

Assumptions

- Components do not share memory.
- Components communicate by asynchronous requests.
- Communications are performed over some bindings.
- It is possible to mark messages.
- It is possible to identify the *safe state* of each component.
- Requirements
  - Absence of deadlocks.
  - Absence of livelocks.
  - Fairness.

Characteristics of the algorithm

- Not specialized for a given configuration.
- Supports replies (futures).
- If the system can’t be stopped safely, the algorithm never finishes.

Agenda

- Introduction and objectives
  - Assumptions
  - Characteristics of the algorithm
- **Description of algorithm**
  - Example
- Conclusion
  - Extension and futures works

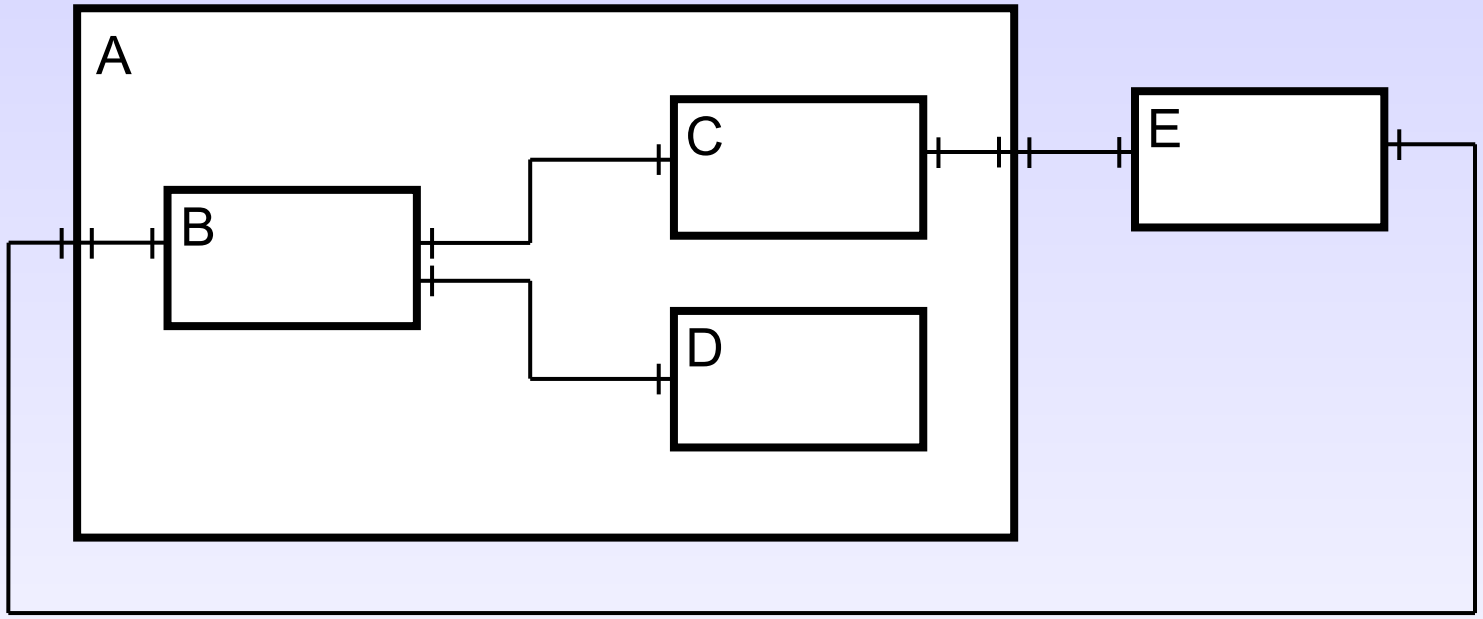
Algorithm description

- The *master component* receives initially the stop request.
- First phase: preparation
  - Only the *master component*.
  - The *master* marks all requests it sends.
- Second phase: stopping system.
  - The *master* blocks the requests it receives.

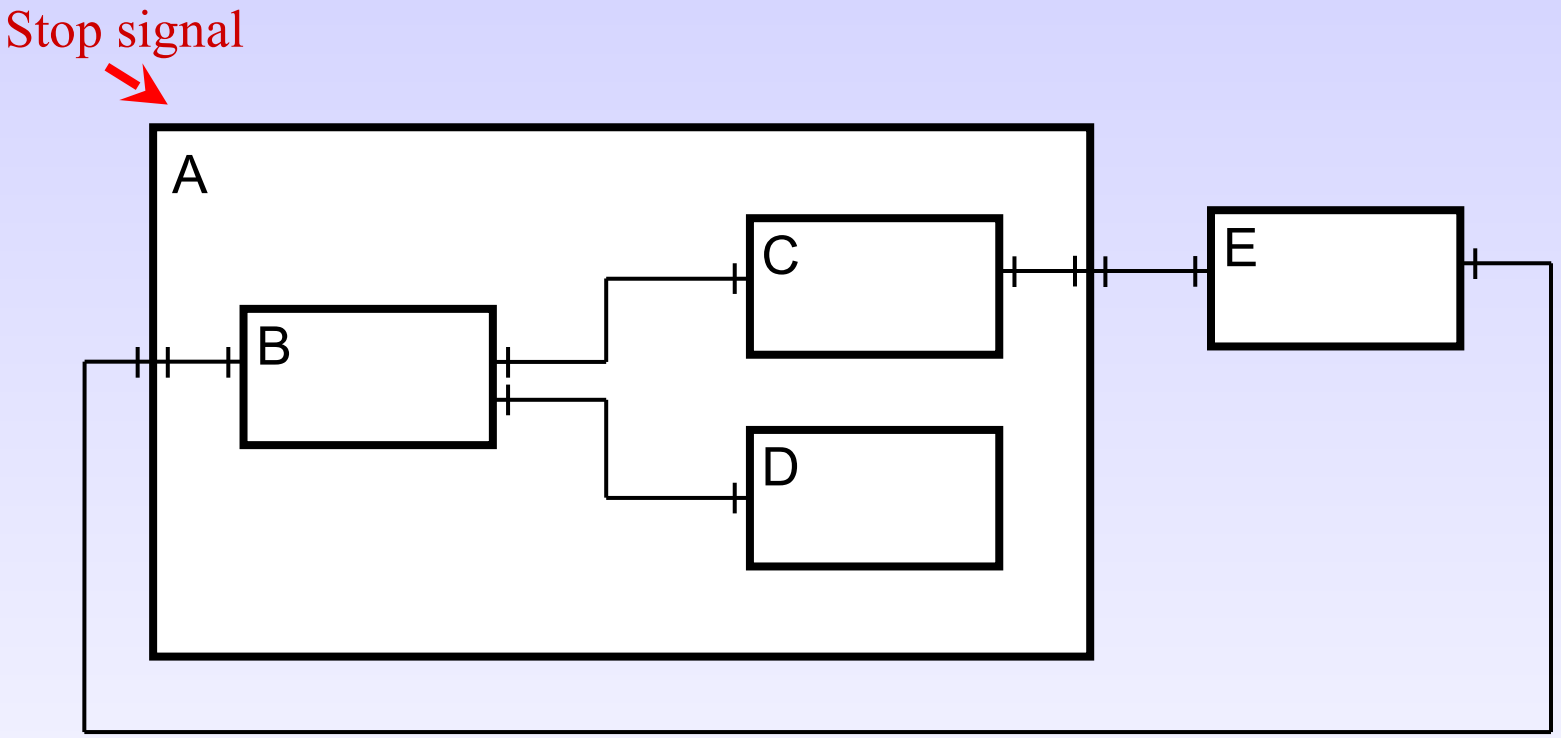
Algorithm description

- States of the components:
  - Started.
  - Wait for stopping (only *master component*).
  - Stopping.
  - Ready to stop. (R2S)
  - Stopped.

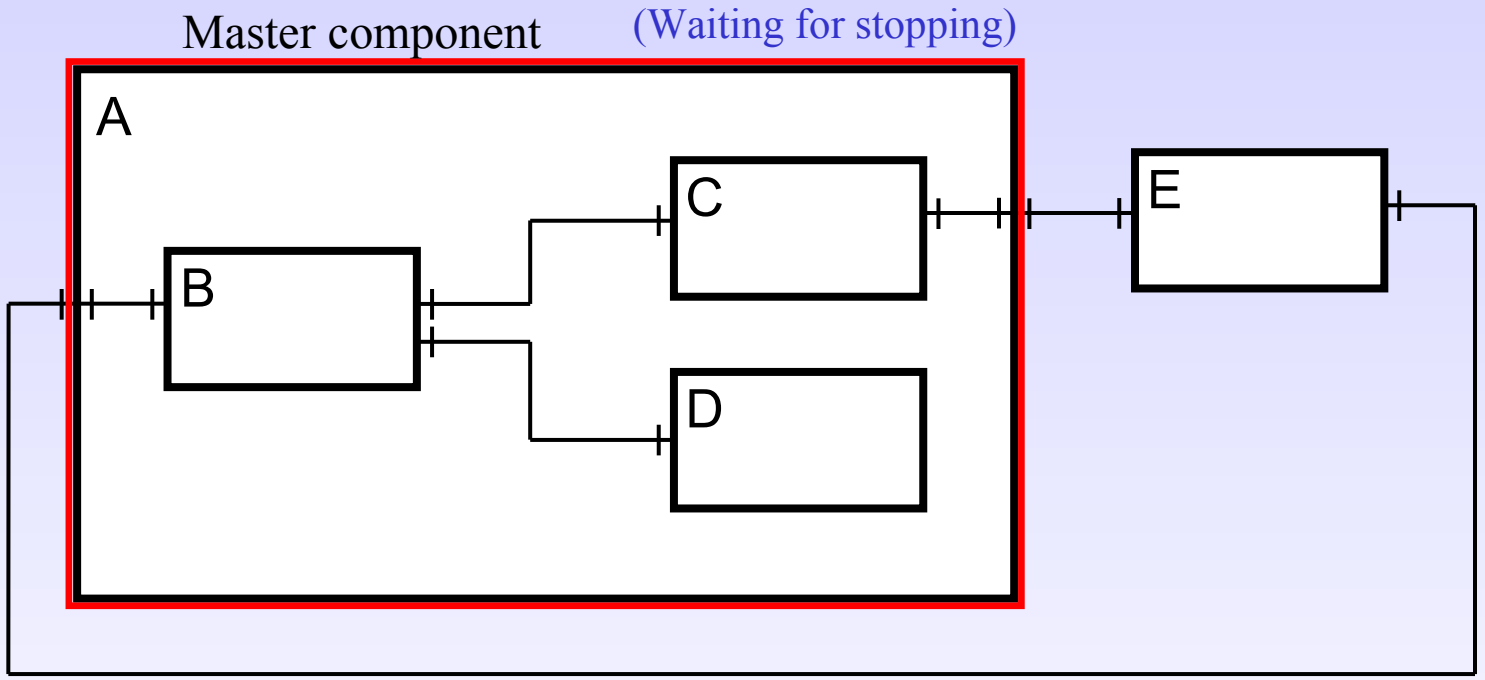
Example



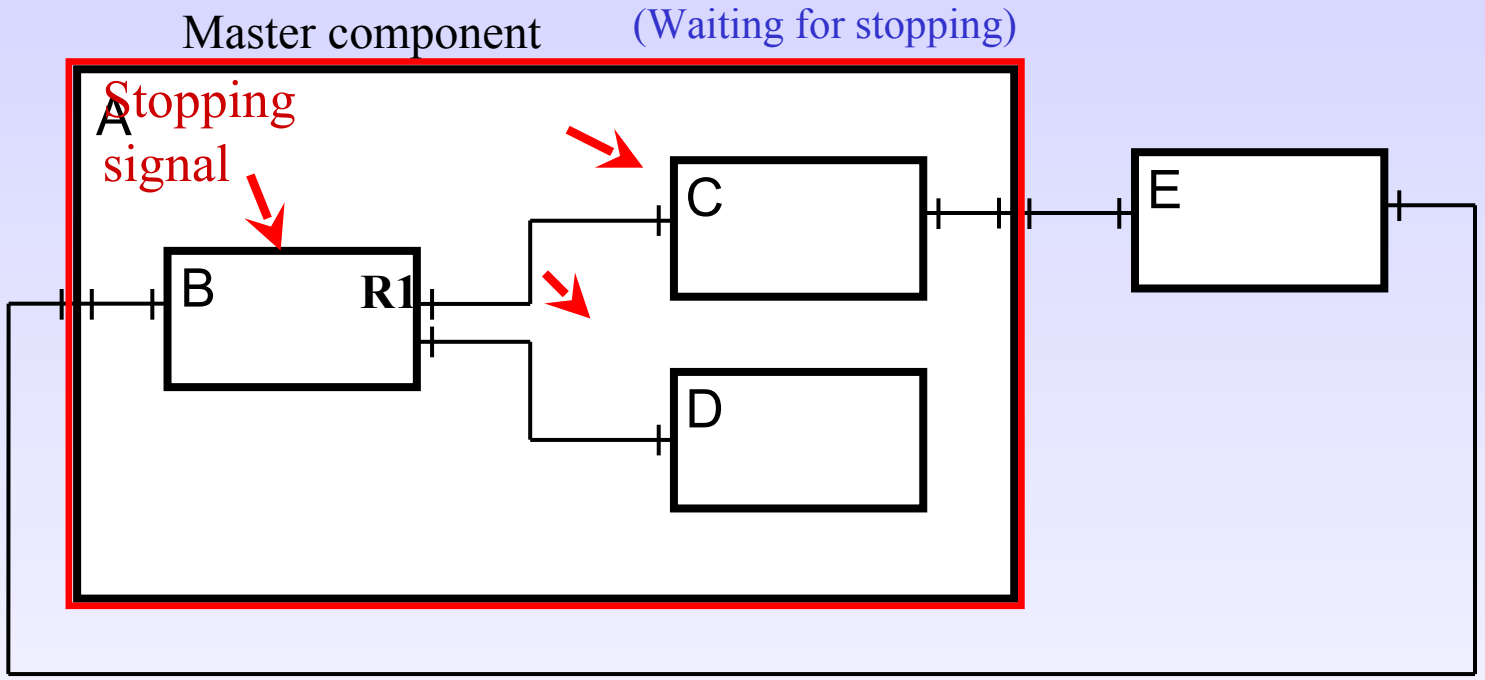
Example



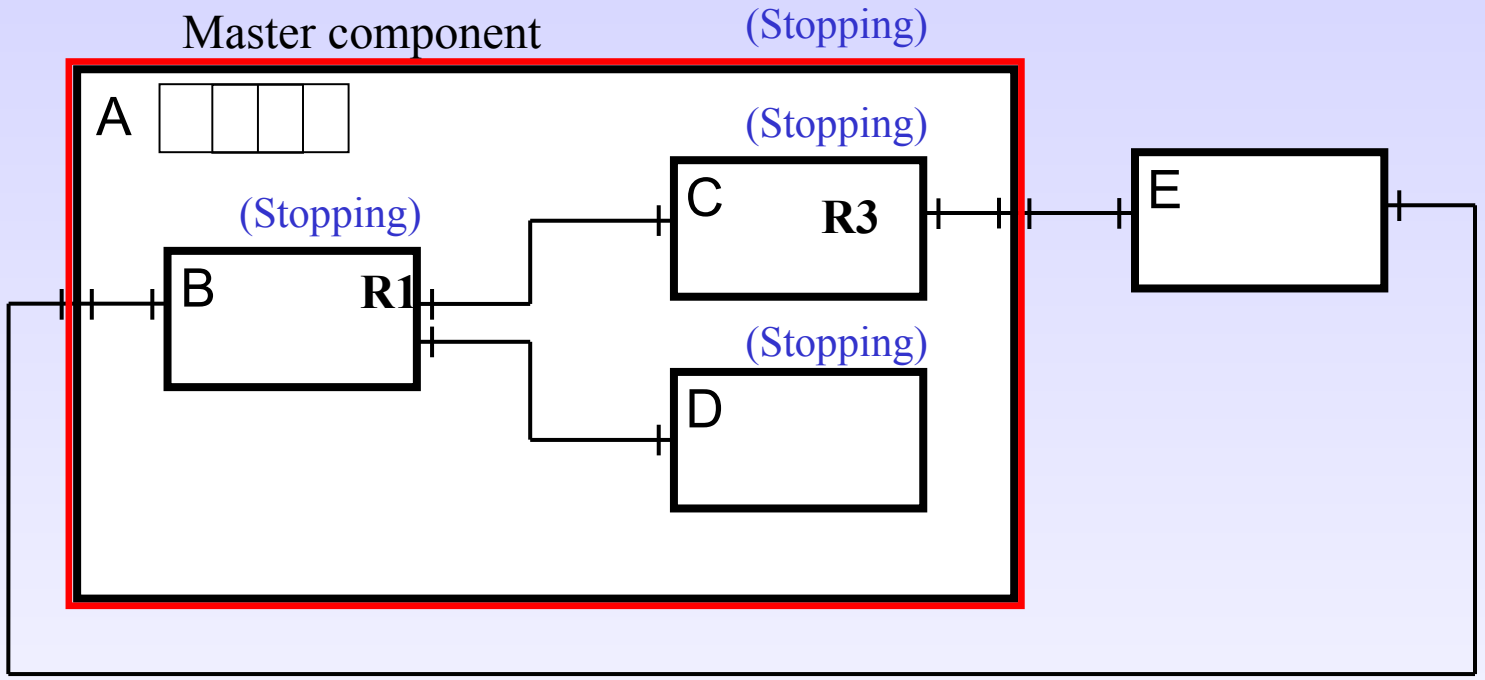
Example



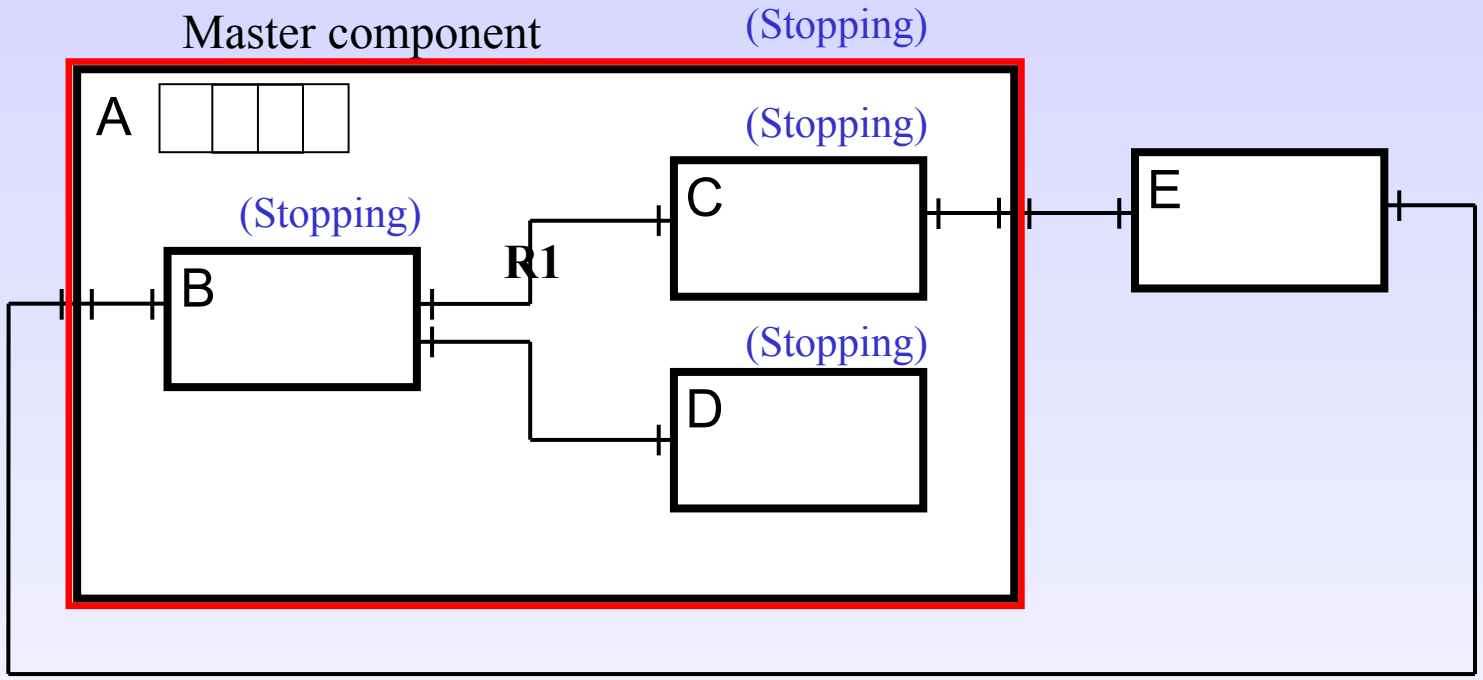
Example


Master component
(Waiting for stopping)
A
Stopping
signal
B
R1
C
D
E

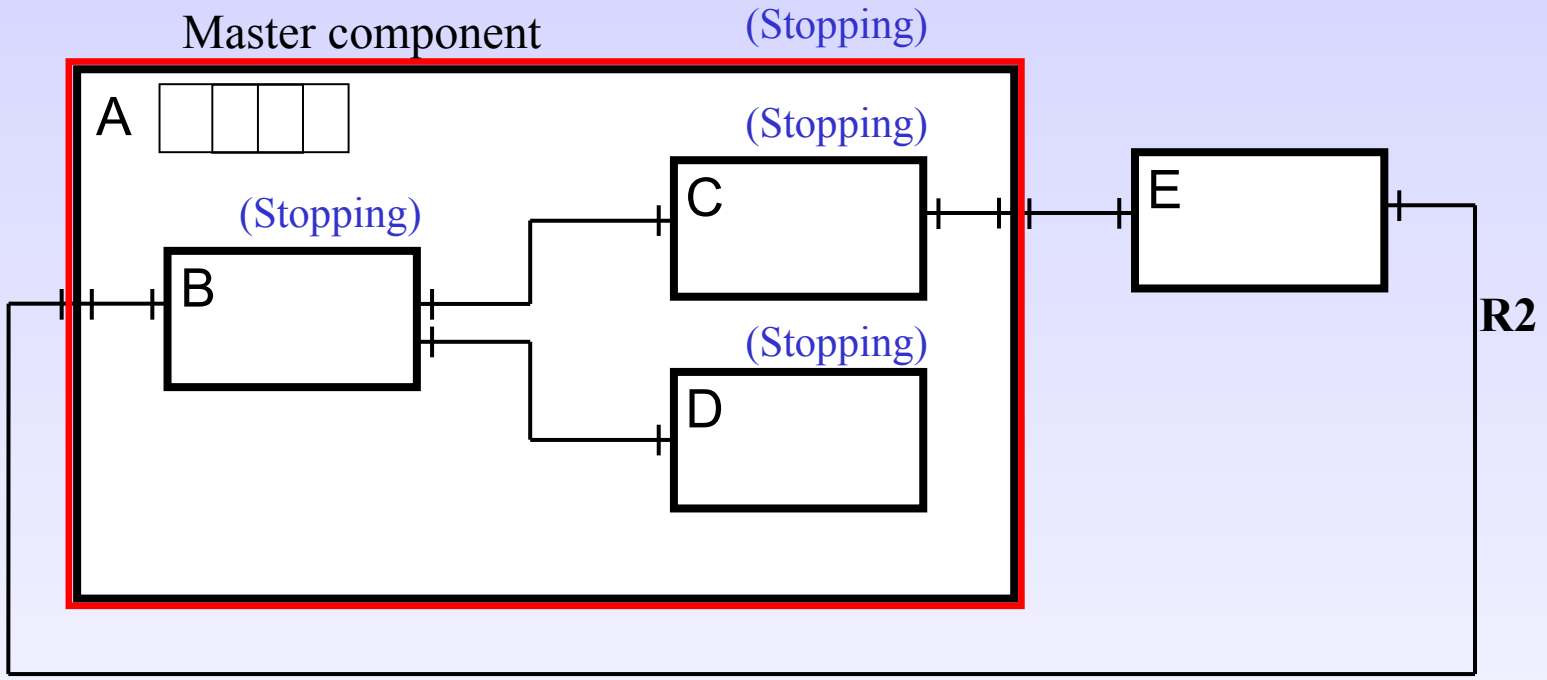
Example


Master component
(Stopping)
A
(Stopping)
C
R3
(Stopping)
B
R1
E
(Stopping)
D

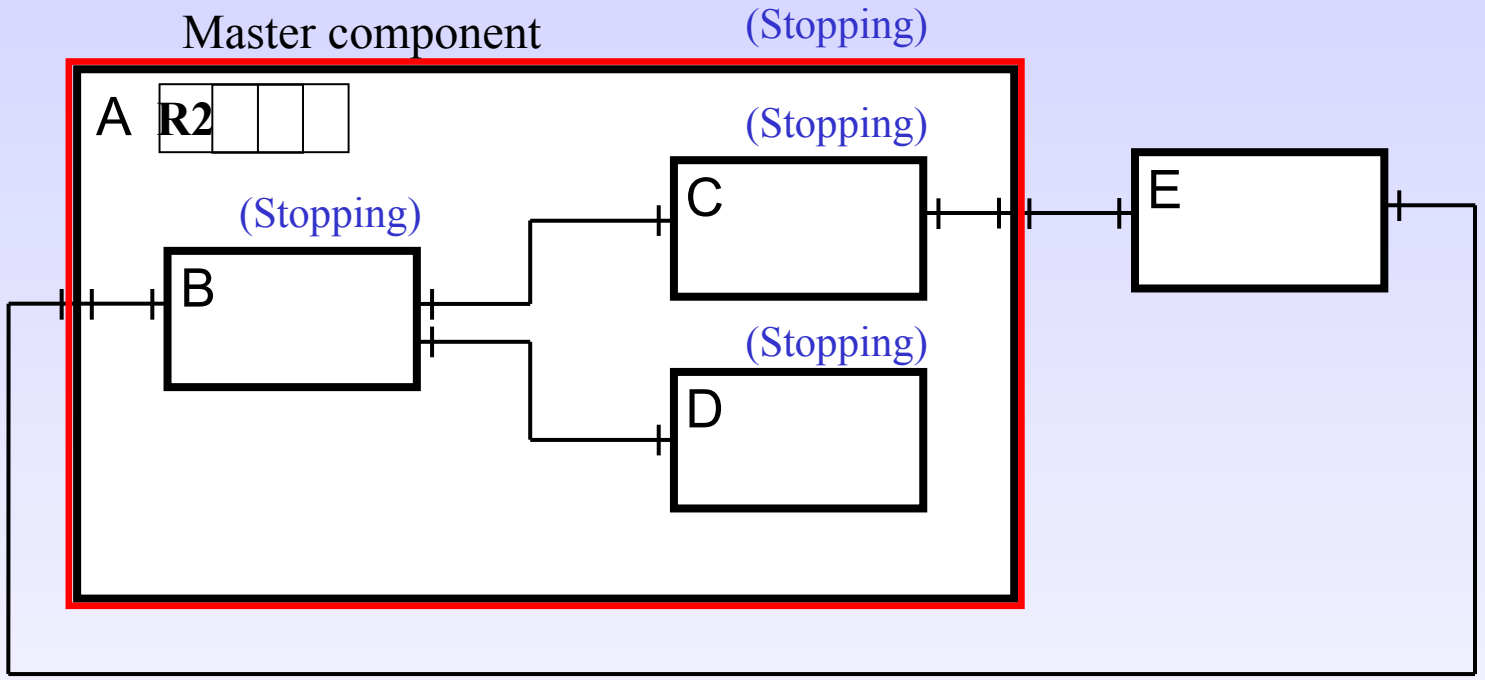
Example


Master component
(Stopping)
A
(Stopping)
(Stopping)
C
E
B
R1
(Stopping)
D

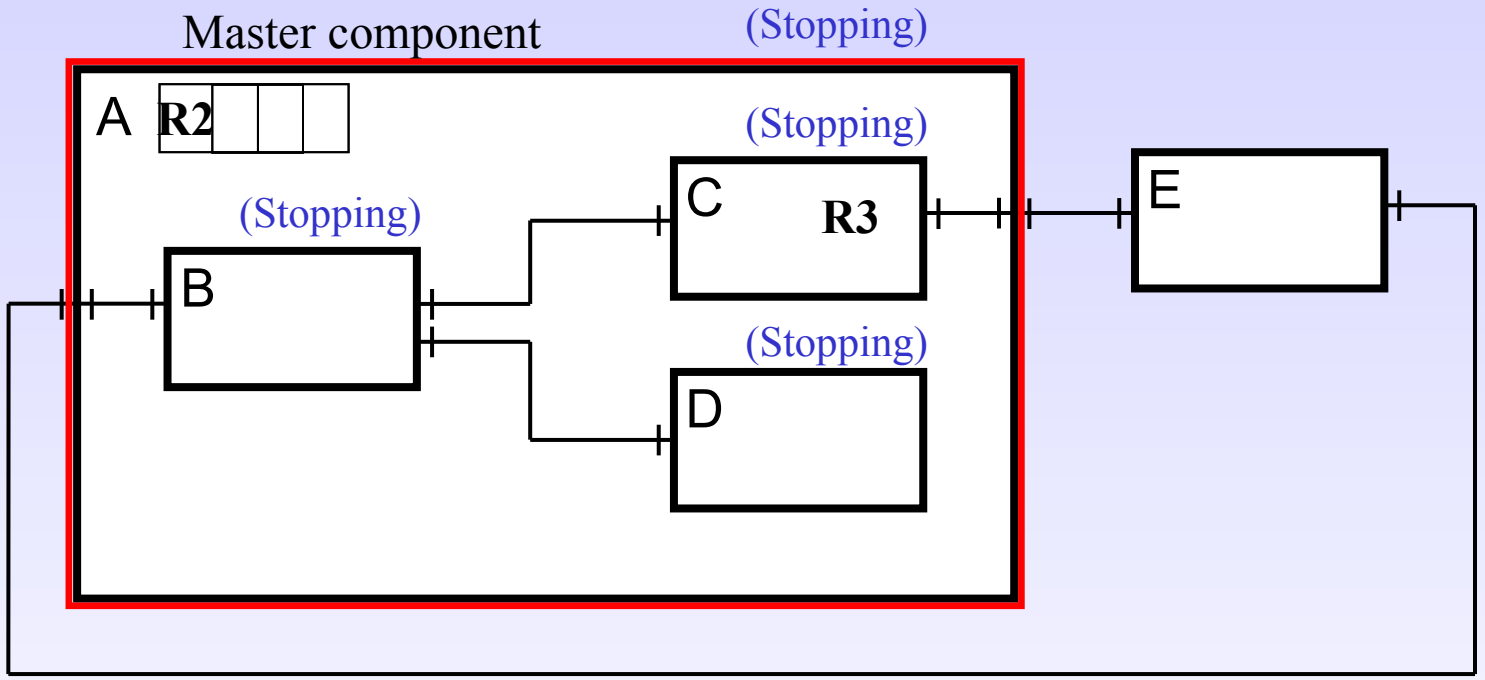
Example


Master component
(Stopping)
A
(Stopping)
(Stopping)
B
C
E
(Stopping)
D
R2

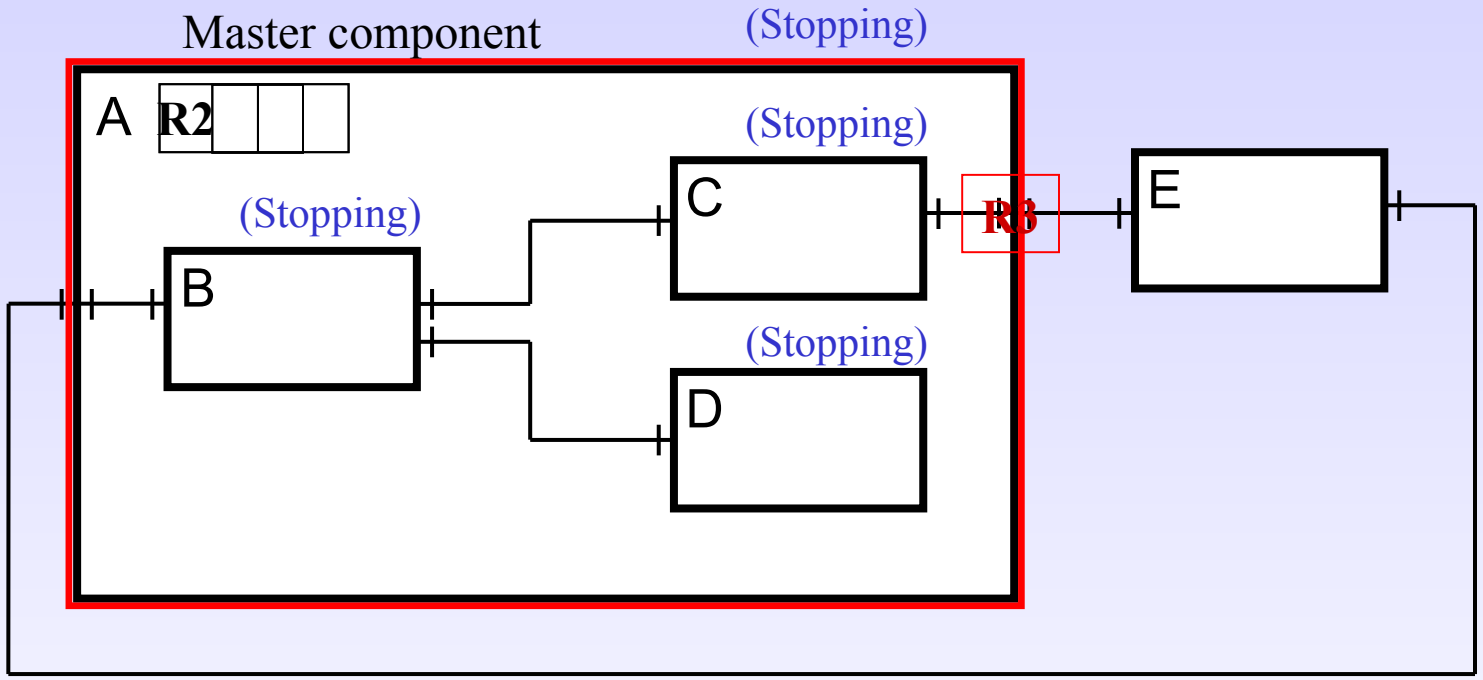
Example


Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
C
E
(Stopping)
D

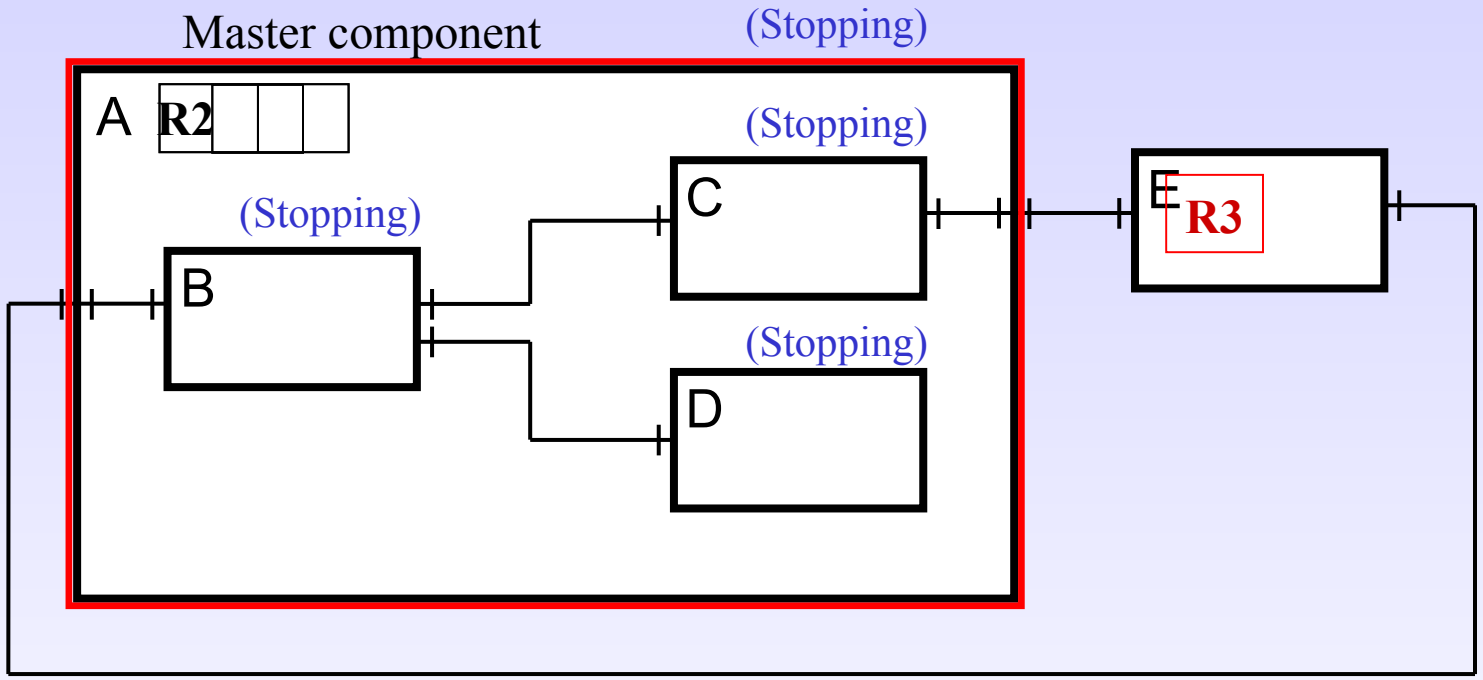
Example


Master component
(Stopping)
A
R2
(Stopping)
C
R3
E
(Stopping)
B
(Stopping)
D

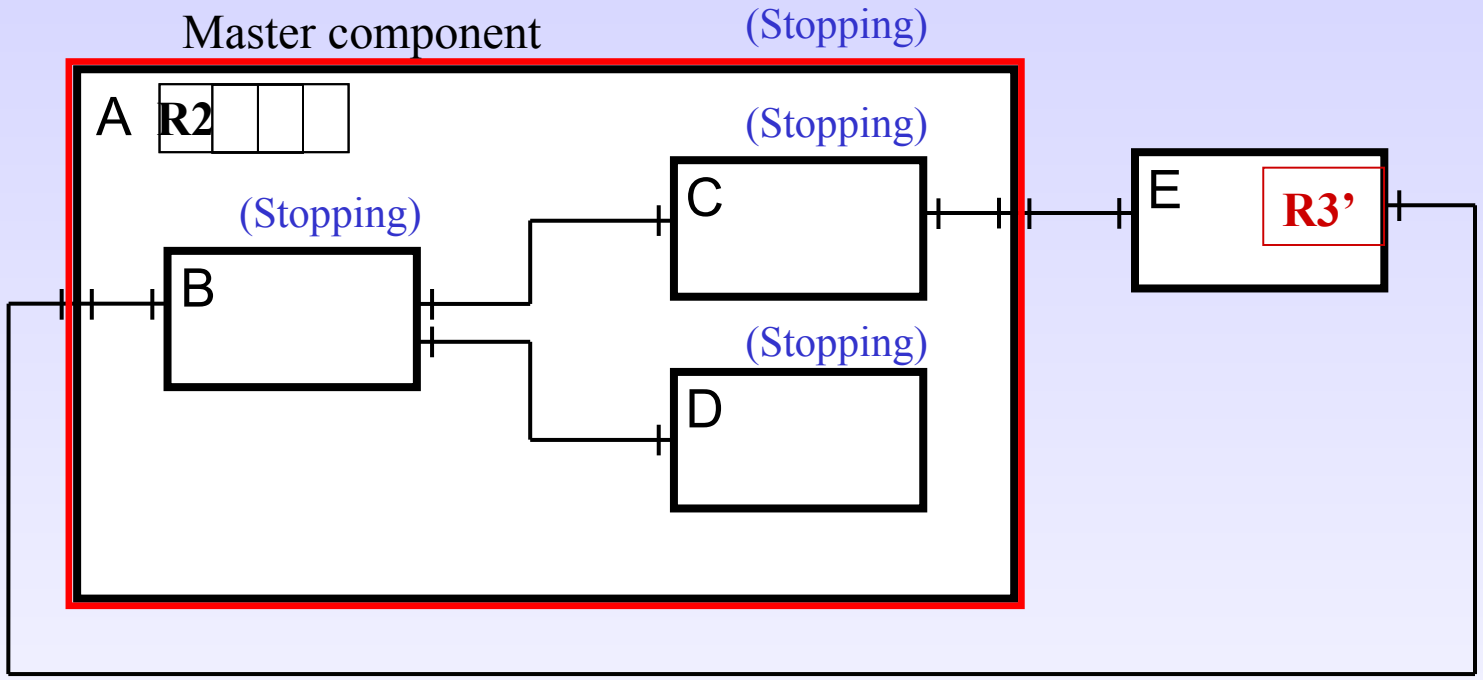
Example


Master component
(Stopping)
A
R2
(Stopping)
C
R3
E
(Stopping)
B
(Stopping)
D

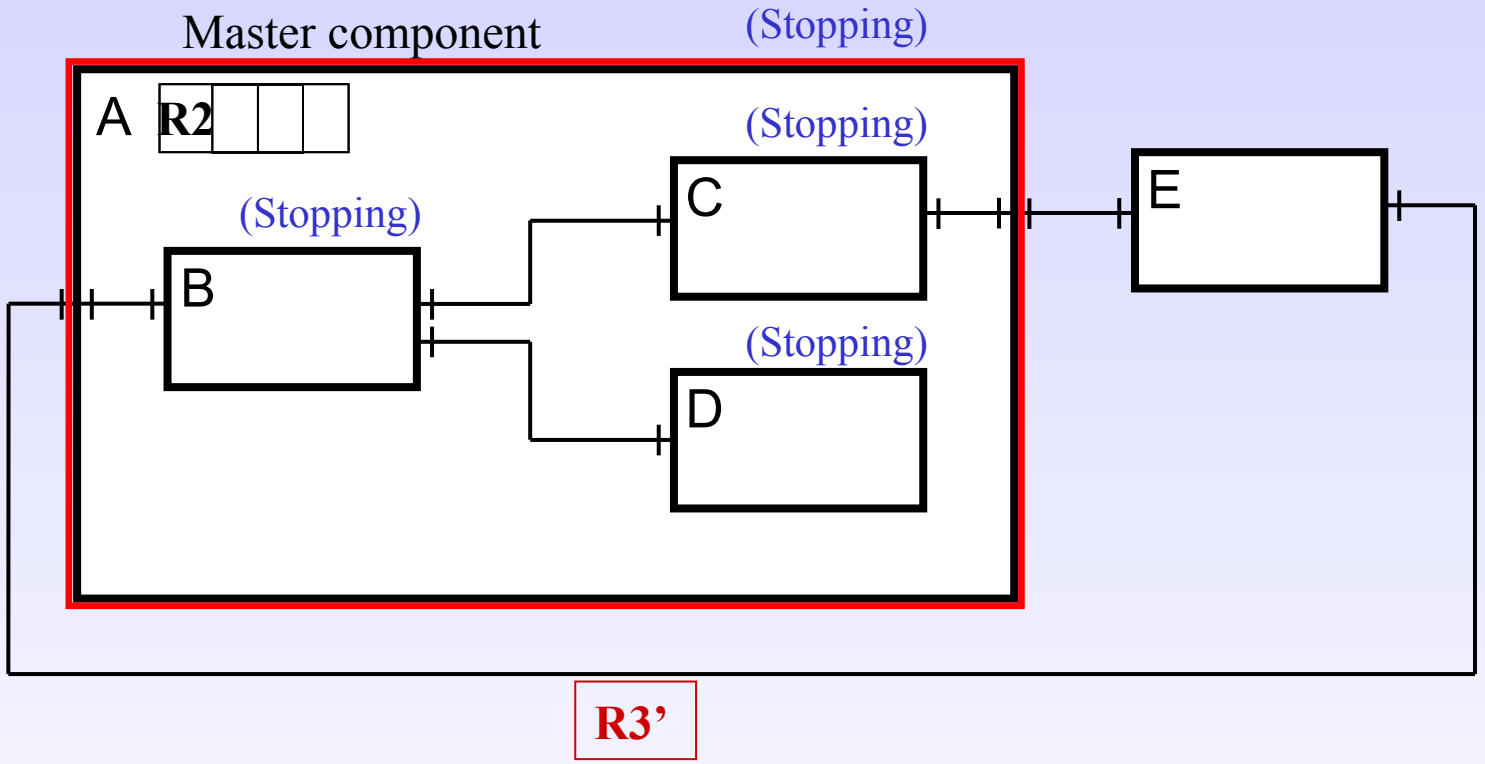
Example


Master component
(Stopping)
A
R2
(Stopping)
C
(Stopping)
B
E
R3
(Stopping)
D

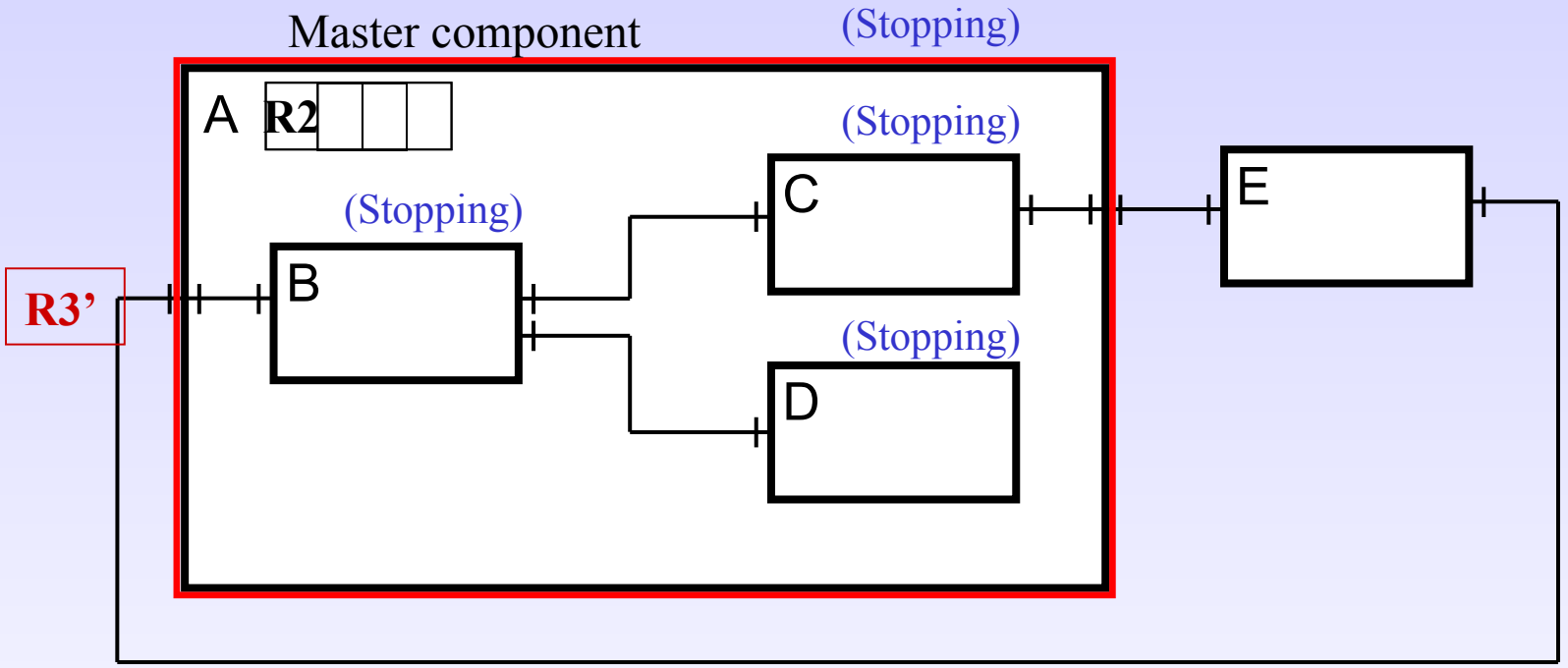
Example


Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
C
E
R3'
(Stopping)
D

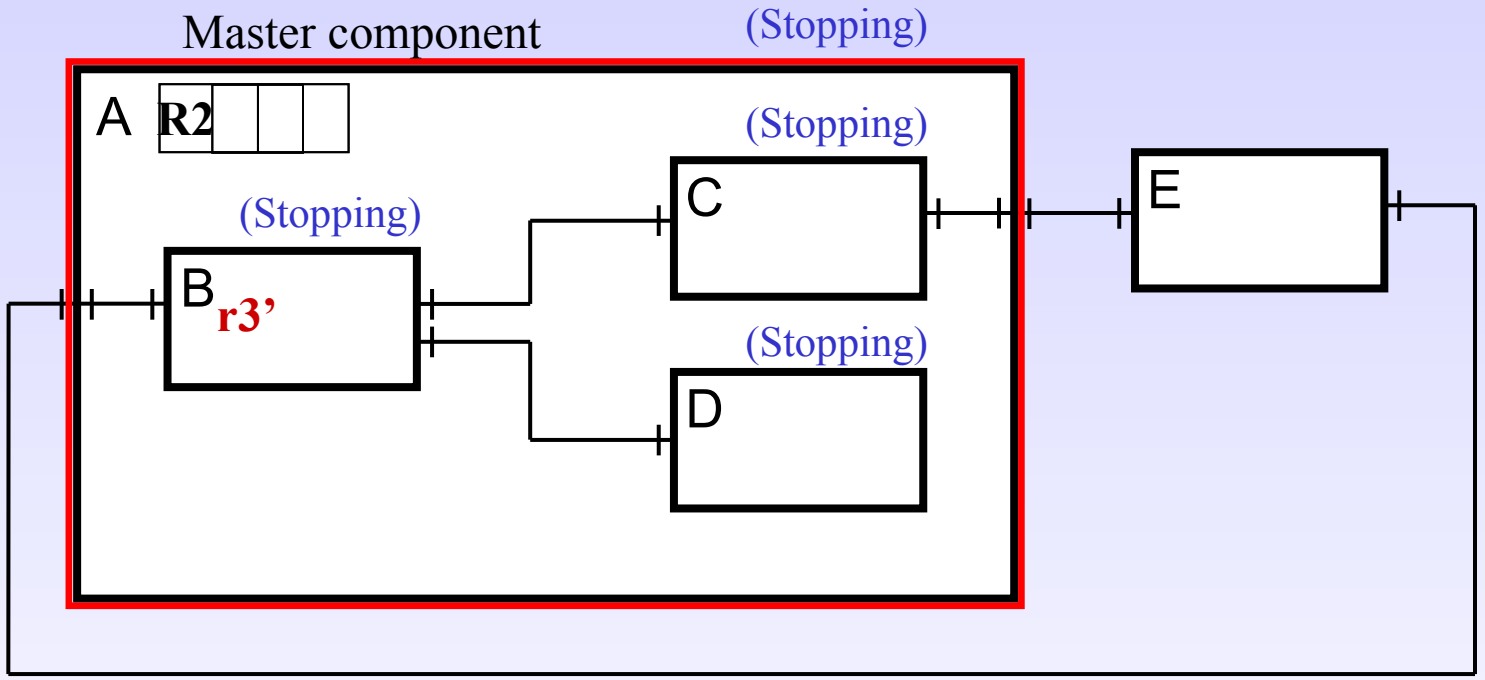
Example


Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
C
E
(Stopping)
D
R3'

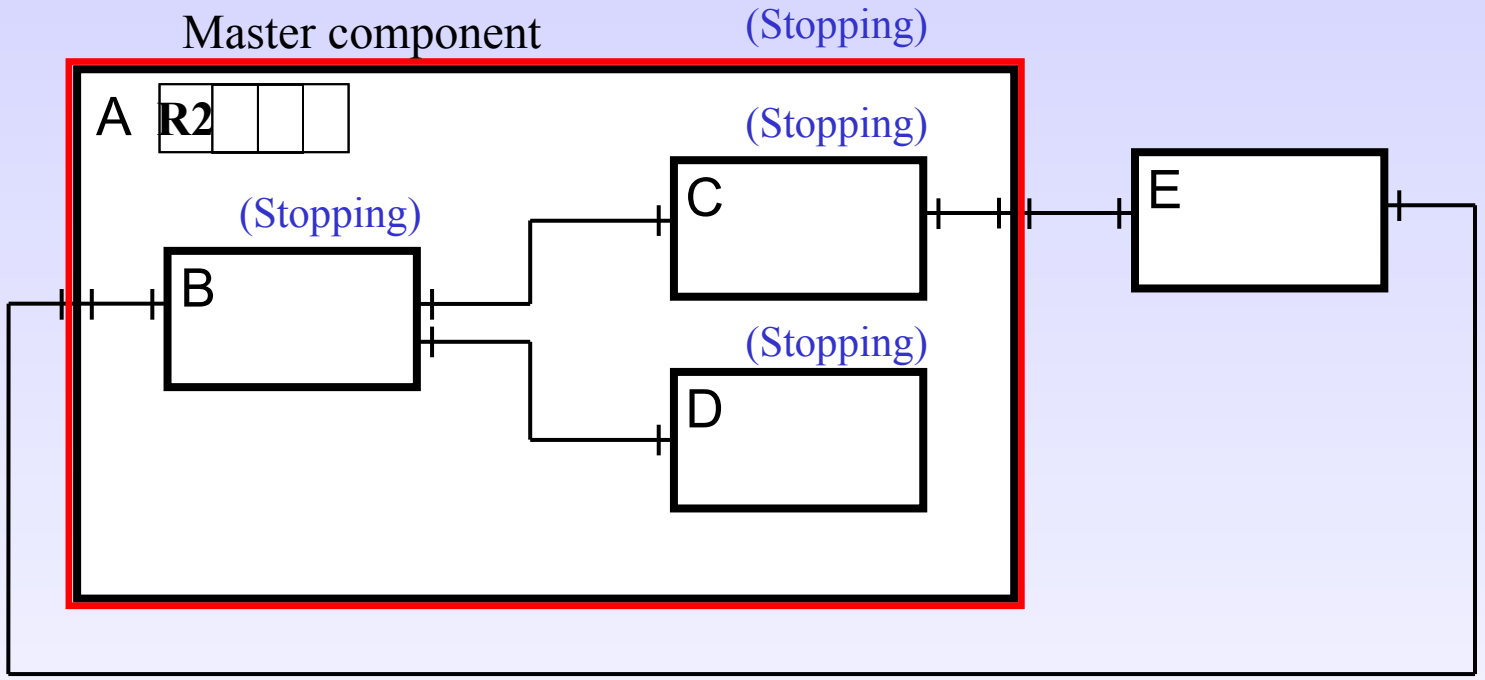
Example


Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
C
E
B
R3'
(Stopping)
D

Example

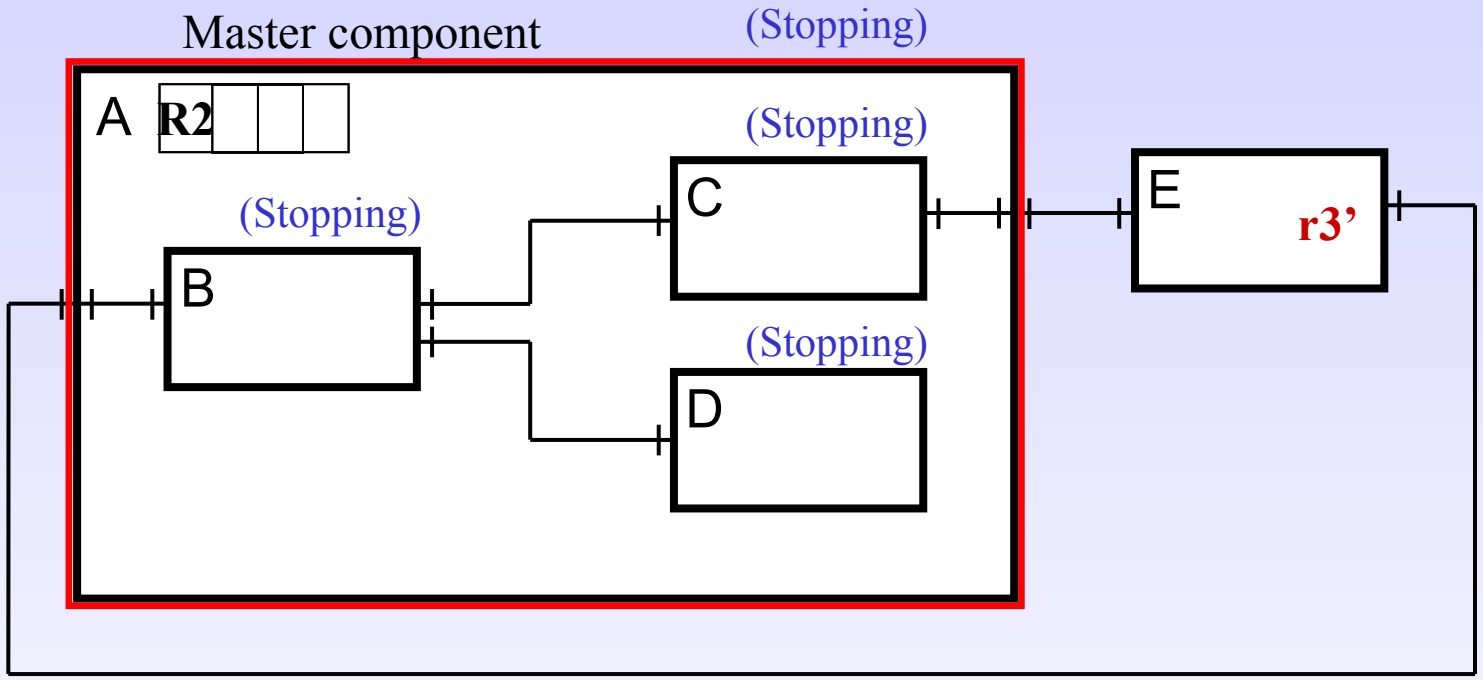

Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
r3'
C
E
(Stopping)
D

Example

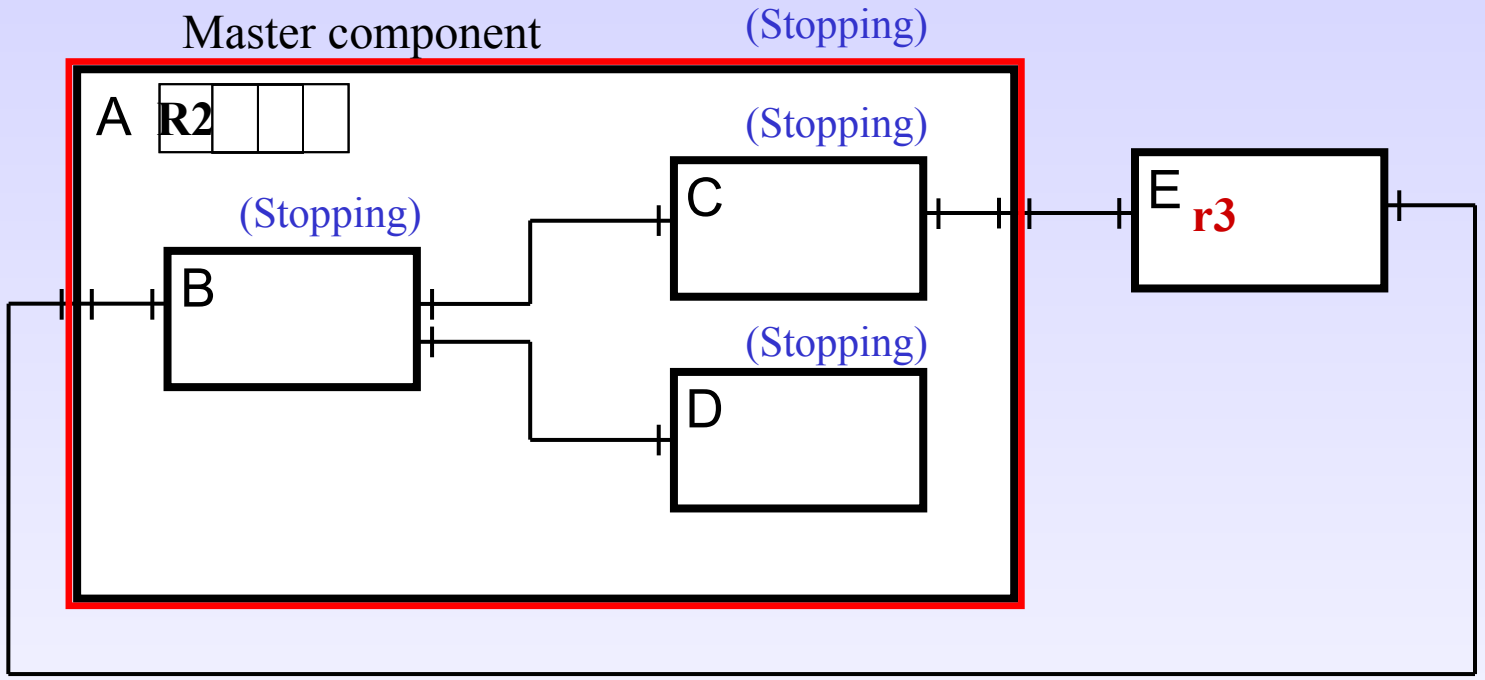

Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
C
E
(Stopping)
D

r3'

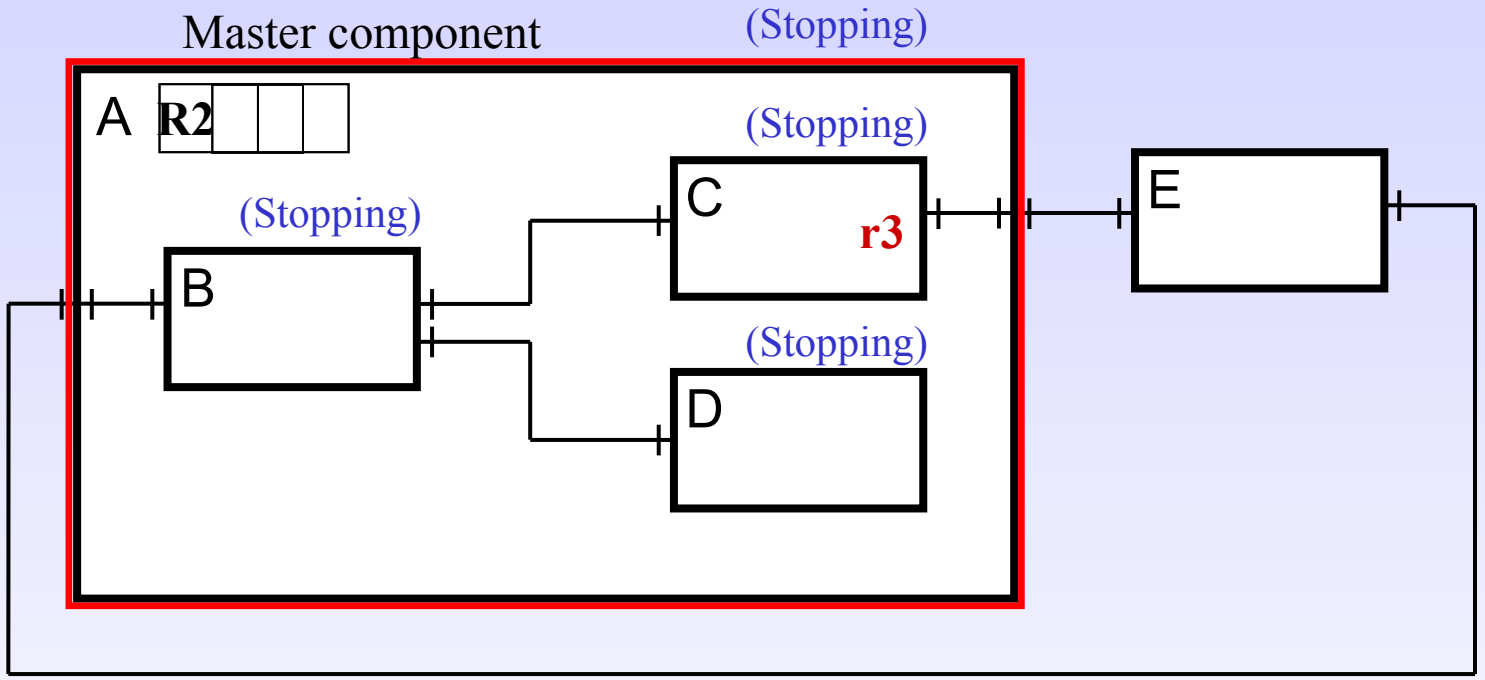
Example


Master component
(Stopping)
A
R2
(Stopping)
C
(Stopping)
B
E
r3'
(Stopping)
D

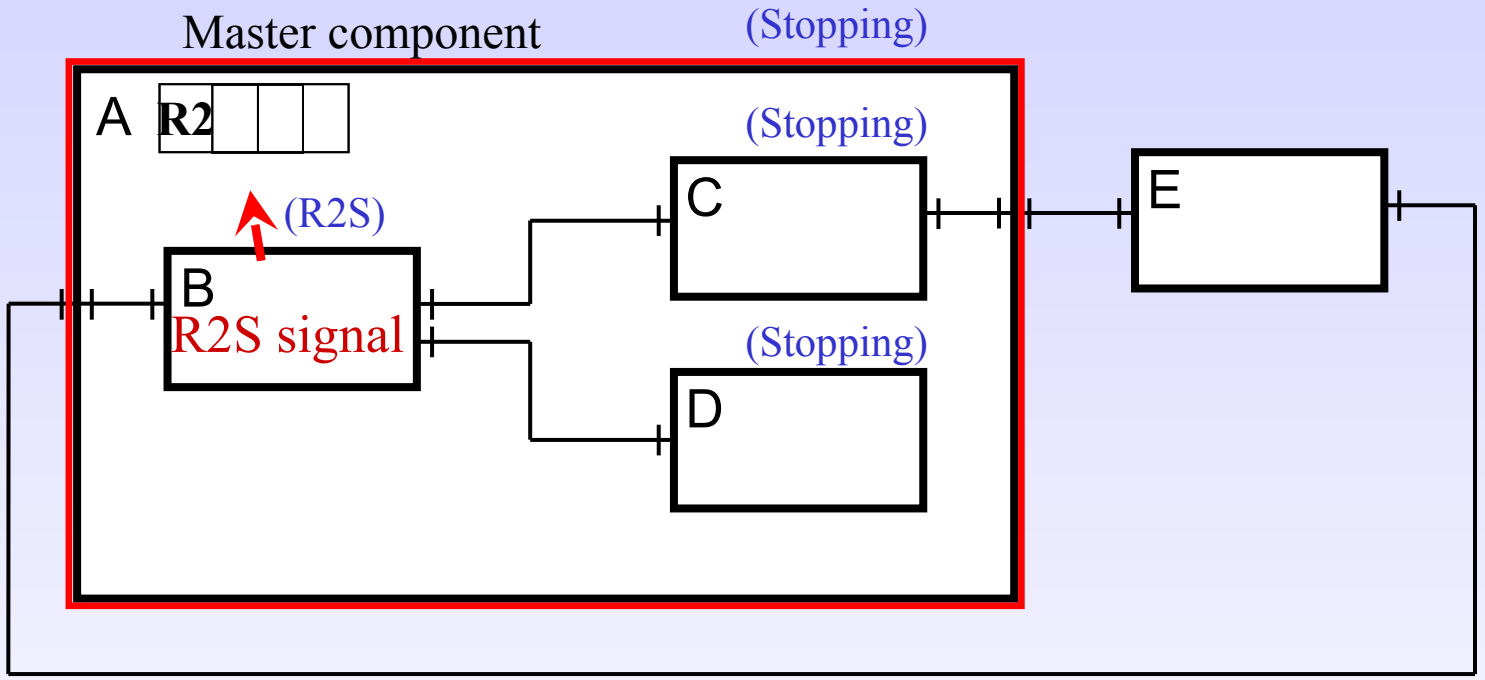
Example


Master component
(Stopping)
A
R2
(Stopping)
(Stopping)
B
C
E
r3
(Stopping)
D

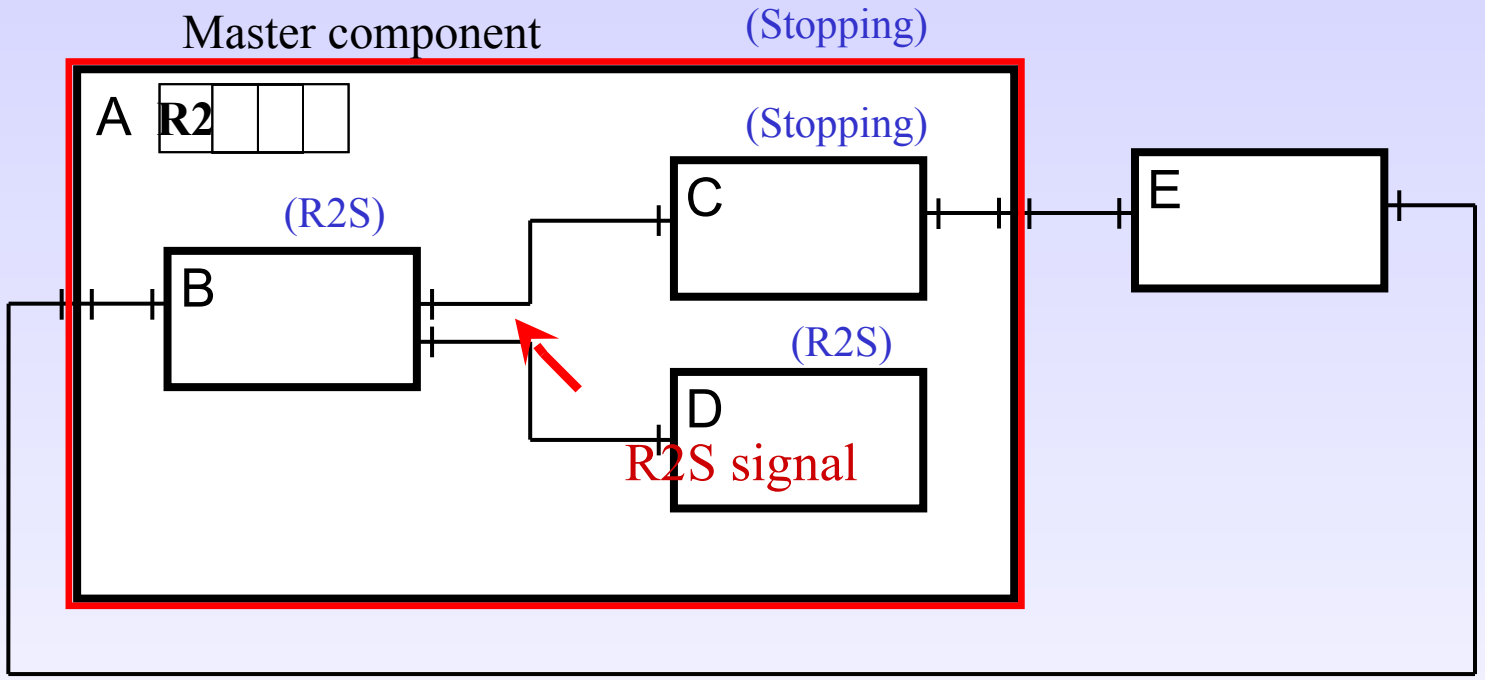
Example


Master component
(Stopping)
A
R2
(Stopping)
C
r3
(Stopping)
B
E
(Stopping)
D

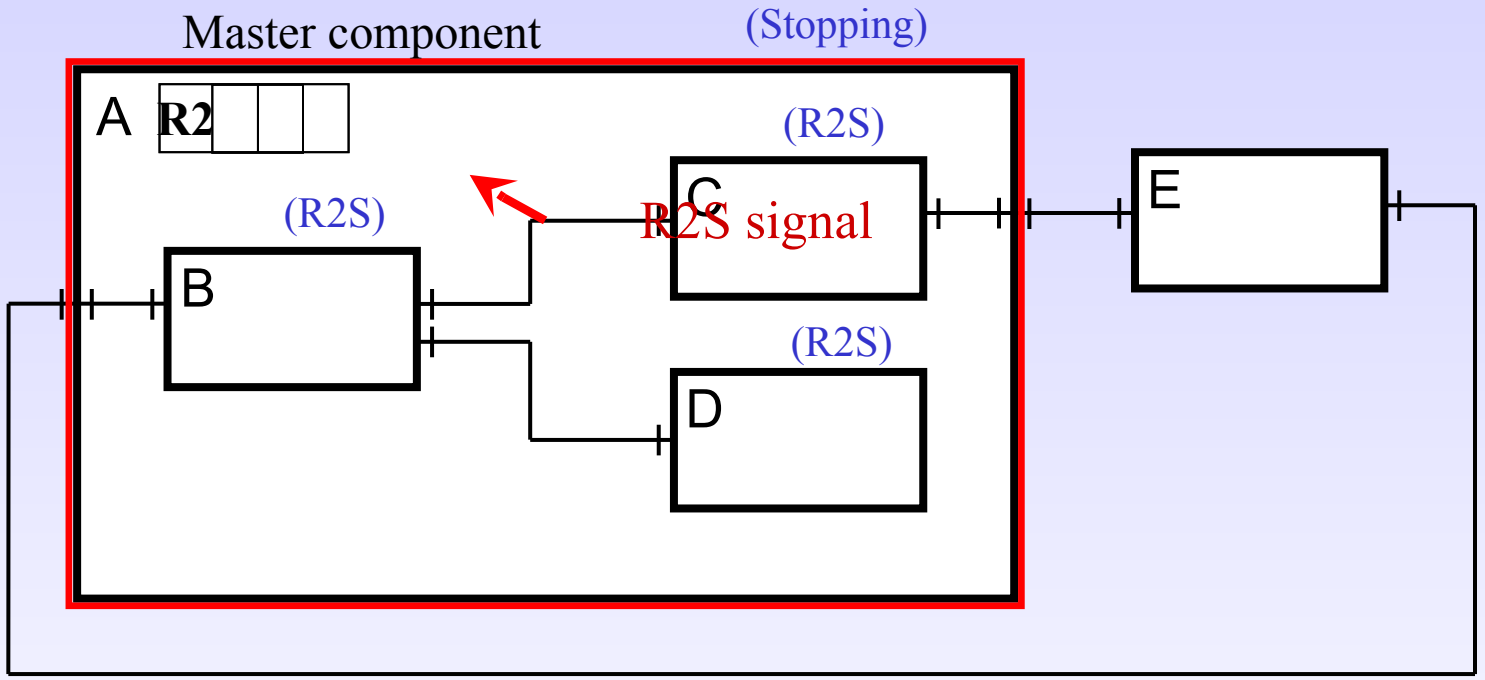
Example


Master component
(Stopping)
A
R2
(Stopping)
(R2S)
C
E
B
R2S signal
(Stopping)
D

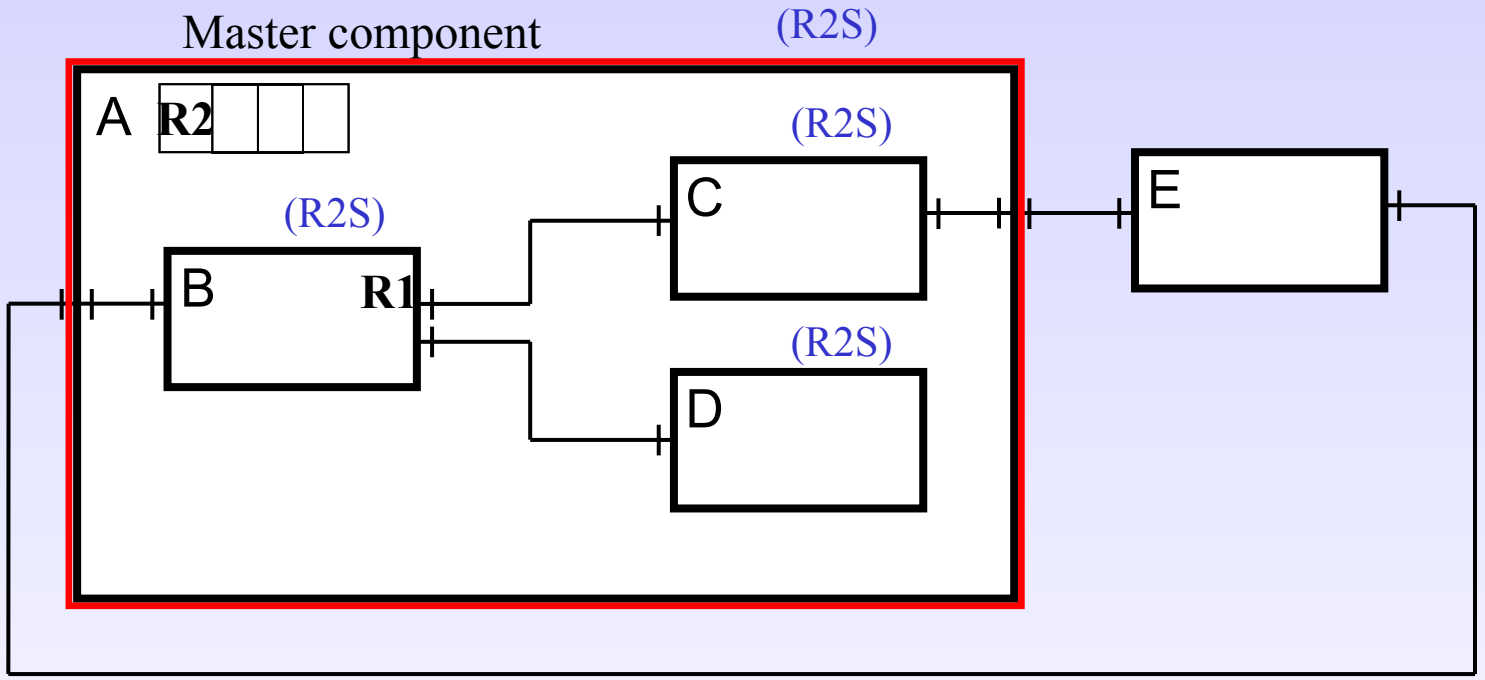
Example


Master component
(Stopping)
A
R2
(Stopping)
(R2S)
C
E
B
(R2S)
D
R2S signal

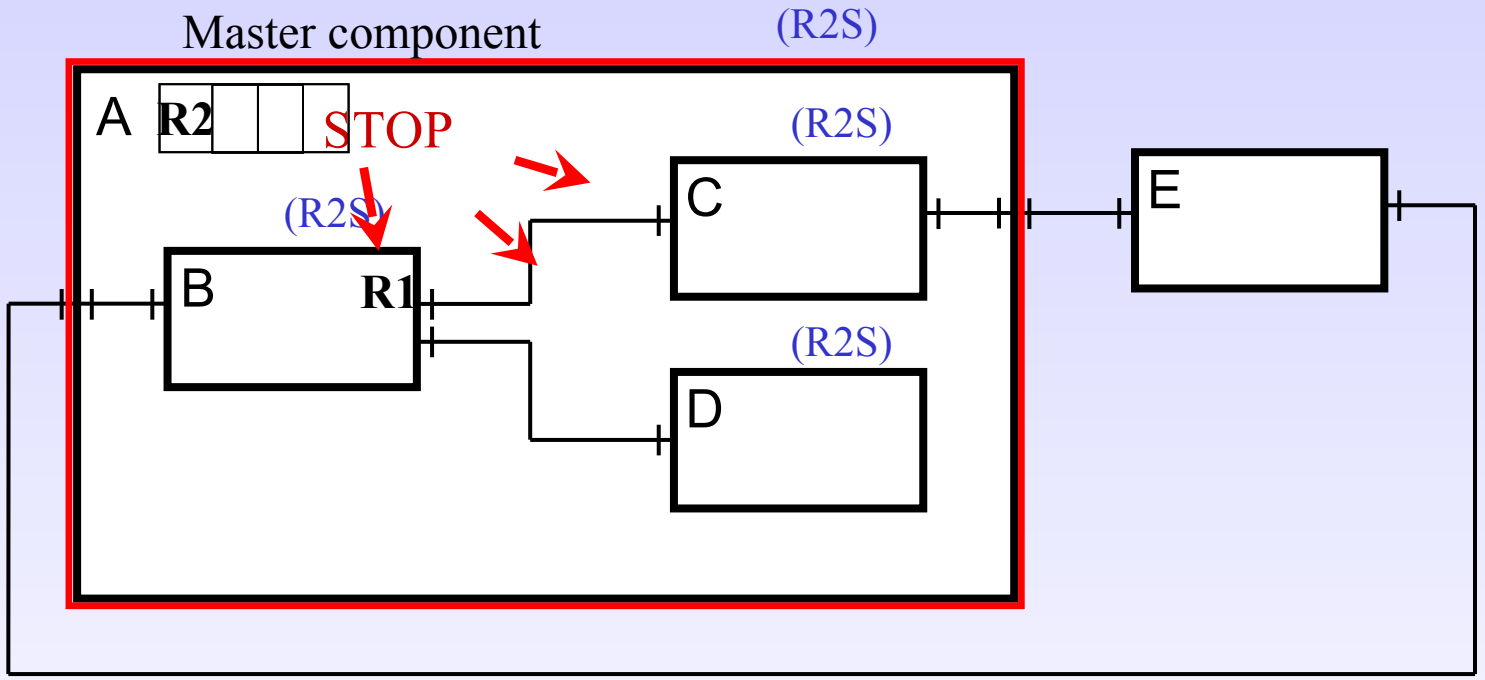
Example



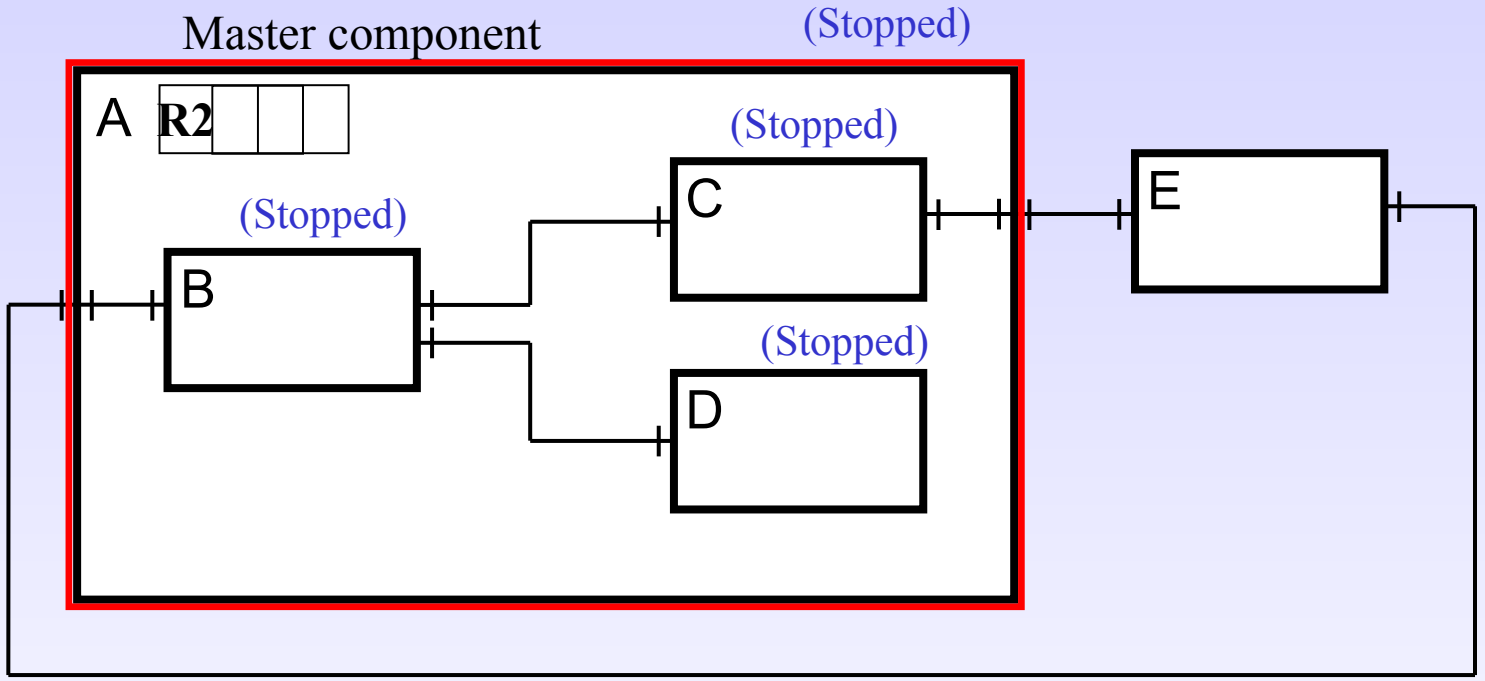
Example



Example



Example


Master component
(Stopped)
A
R2
(Stopped)
C
E
(Stopped)
B
(Stopped)
D

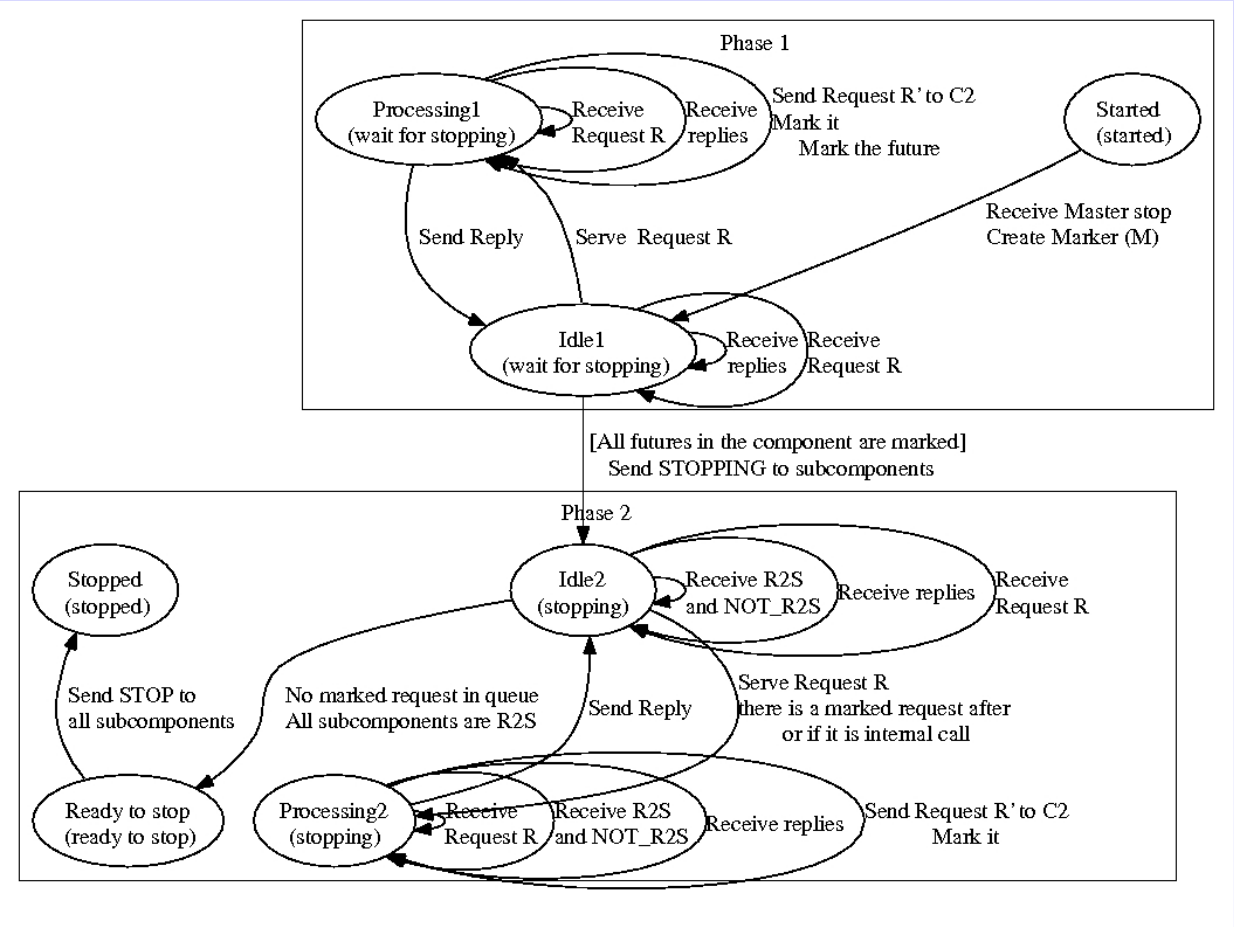
Algorithm description

- Initially all components are in *started state.*
- The *master component* changes to *waiting for stopping state* when it receives the stop request.
- While the master is in *waiting for stopping state* it marks all outgoing requests.
- When all requests are market the *master component* changes to *stopping state* and it sends the *stopping signal* to its subcomponents.

Algorithm description

- When the subcomponents receive the *stopping signal* they change to stopping state.
- When there are no more internal calls to process, each components send a *R2S signal* and change to *R2S state*.
- When the *master component* receives the *R2S signal* it sends the *stop signal* for stopped the system.

Algorithm for the master component



Agenda

- Introduction and objectives
  - Assumptions
  - Characteristics of the algorithm
- Description of algorithm
  - Example
- **Conclusion**
  - Extension and futures works

Conclusion

- Any component without deadlock and livelock is safely stopped:
  - The master component is in *strongly idle state (quiescent)*.
  - Master queue only contains requests received after the last served request.
- The algorithm is independent of components topology or components behavior.
- Our contribution can be considered as an adaptation and an extension of the lifecycle controller of Fractal for GCM.
- Prototype implementation in GCM.

Extension and future works

- Stopping several components.
  - Marking states and requests with the identifier of the master.
- Non hierarchical systems
  - One central manager.
  - Intercepts calls outgoing the assembly.
  - Intercepts calls ingoing the assembly.
- Verify formally the stopping algorithm.
- Considering hierarchical systems with shared components.

Thanks