

# Eden : a Consensus Based Group Communication System

Frédéric Tronel

ftronel@irisa.fr

IRISA - INRIA Rennes  
Campus de Beaulieu  
Rennes, France

# Eden Overview

# Eden Overview

- Eden is a Group Communication Services (GCS)

# Eden Overview

- Eden is a Group Communication Services (GCS)
- It has been developed with fault-tolerance in mind, and to be integrated/coupled with a commercial ORB.

# Eden Overview

- Eden is a Group Communication Services (GCS)
- It has been developed with fault-tolerance in mind, and to be integrated/coupled with a commercial ORB.
- It has been developed using Java 1.2, and is divided into two main packages :

# Eden Overview

- Eden is a Group Communication Services (GCS)
- It has been developed with fault-tolerance in mind, and to be integrated/coupled with a commercial ORB.
- It has been developed using Java 1.2, and is divided into two main packages :
  - Eva : an event based architecture for building high level protocols

# Eden Overview

- Eden is a Group Communication Services (GCS)
- It has been developed with fault-tolerance in mind, and to be integrated/coupled with a commercial ORB.
- It has been developed using Java 1.2, and is divided into two main packages :
  - Eva : an event based architecture for building high level protocols
  - Adam : the GCS package itself that relies on an Generic Agreement Framework (GAF)



# Eva an Event Based Architecture for Building High Level Protocol

# Motivations

# Motivations

- Building distributed algorithms is a difficult task.

# Motivations

## Task 1

**send (PROPOSITION $\langle p_i, r_i, est_i \rangle$ ) to all;**

**upon receipt of PROPOSITION $\langle p_j, r_j, est_j \rangle$ )**

## Task 2

**wait (50 ms);**

- Building distributed algorithms is a difficult task.
- Most of the protocols described in the literature, are written with a reactive model in mind.

# Motivations

## Task 1

**send** (PROPOSITION $\langle p_i, r_i, est_i \rangle$ ) **to** all;

**upon receipt of** PROPOSITION $\langle p_j, r_j, est_j \rangle$ )

## Task 2

**wait** (50 ms);

- Building distributed algorithms is a difficult task.
- Most of the protocols described in the literature, are written with a reactive model in mind.
  - The algorithm is decomposed into autonomous tasks

# Motivations

## Task 1

**send** (PROPOSITION $\langle p_i, r_i, est_i \rangle$ ) **to** all;

**upon receipt of** PROPOSITION $\langle p_j, r_j, est_j \rangle$ )

## Task 2

**wait** (50 ms);

- Building distributed algorithms is a difficult task.
- Most of the protocols described in the literature, are written with a reactive model in mind.
  - The algorithm is decomposed into autonomous tasks
  - Each task can **send** messages

# Motivations

## Task 1

**send** (PROPOSITION $\langle p_i, r_i, est_i \rangle$ ) **to** all;

**upon receipt of** PROPOSITION $\langle p_j, r_j, est_j \rangle$ )

## Task 2

**wait** (50 ms);

- Building distributed algorithms is a difficult task.
- Most of the protocols described in the literature, are written with a reactive model in mind.
  - The algorithm is decomposed into autonomous tasks
  - Each task can send messages, **react** to the **receipt** of a message

# Motivations

## Task 1

**send** (PROPOSITION $\langle p_i, r_i, est_i \rangle$ ) **to** all;

**upon receipt of** PROPOSITION $\langle p_j, r_j, est_j \rangle$ )

## Task 2

**wait** (50 ms);

- Building distributed algorithms is a difficult task.
- Most of the protocols described in the literature, are written with a reactive model in mind.
  - The algorithm is decomposed into autonomous tasks
  - Each task can send messages, **react** to the receipt of a message, the **expiry** of a timeout.

# Eva Main Features Overview

# Eva Main Features Overview

- Event Based Architecture

# Eva Main Features Overview

- Event Based Architecture
- This mimics the way protocol are written

# Eva Main Features Overview

- Event Based Architecture
- This mimics the way protocol are written
- Efficiency obtained by the following technologies :

# Eva Main Features Overview

- Event Based Architecture
- This mimics the way protocol are written
- Efficiency obtained by the following technologies :
  - Events are managed through the use of factories.

# Eva Main Features Overview

- Event Based Architecture
- This mimics the way protocol are written
- Efficiency obtained by the following technologies :
  - Events are managed through the use of factories.
  - Serialization/deserialization of events is tightly coupled with factories.

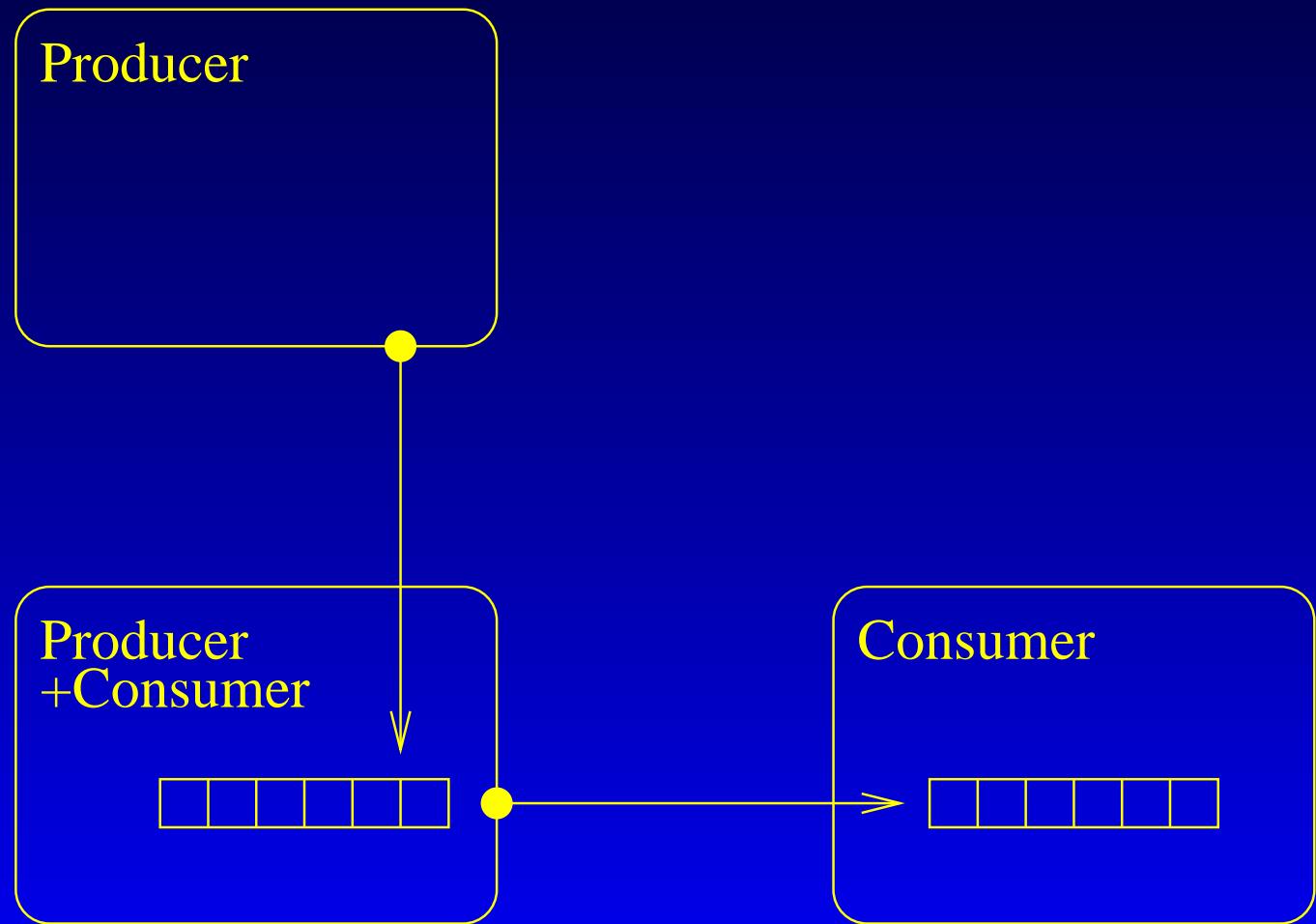
# Eva Main Features Overview

- Event Based Architecture
- This mimics the way protocol are written
- Efficiency obtained by the following technologies :
  - Events are managed through the use of factories.
  - Serialization/deserialization of events is tightly coupled with factories.
  - Eva supports the partial deserialization of events in order to test their freshness.  
Network delayed events are quickly garbaged.

# Event Bus Paradigm

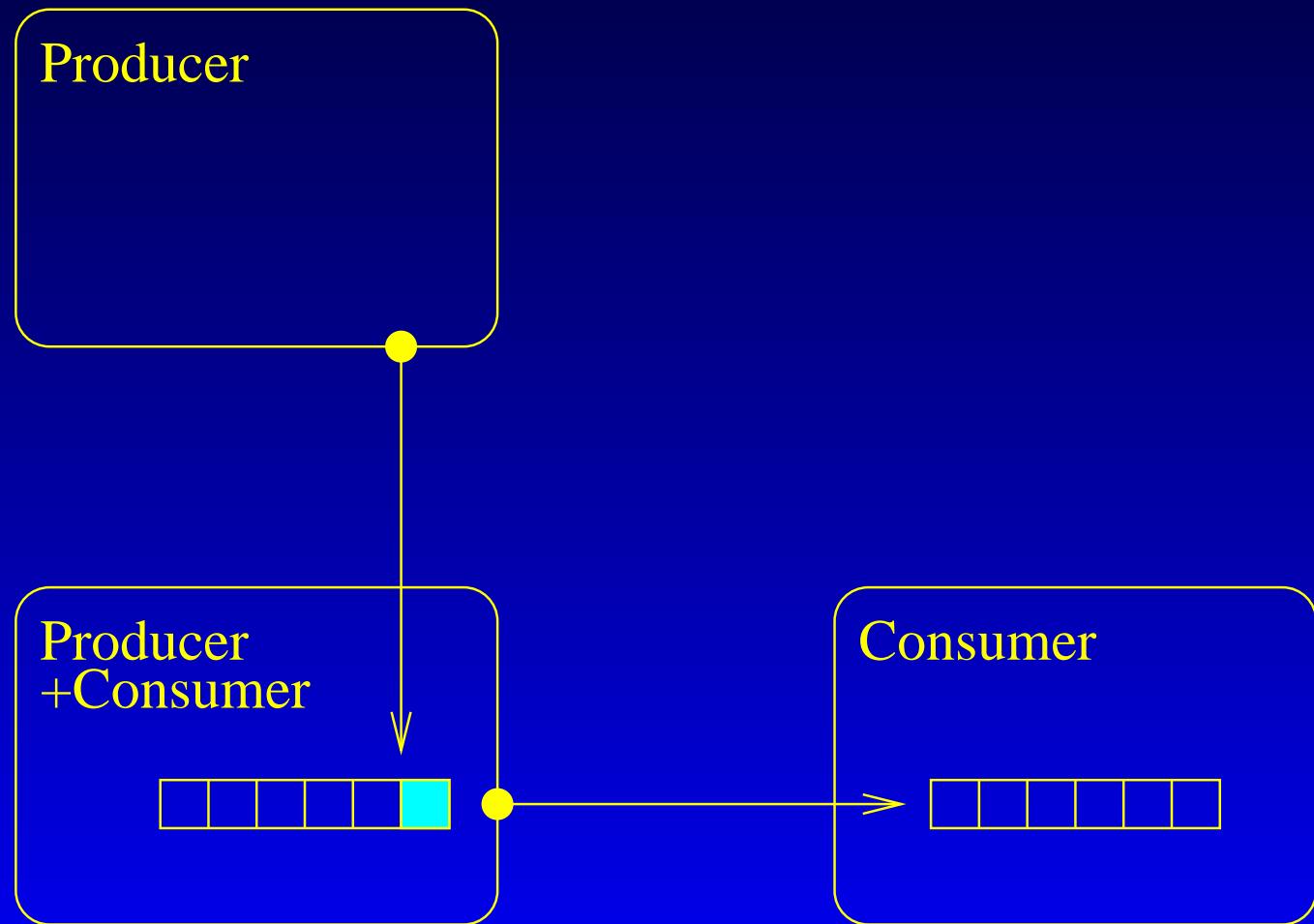
# Event Bus Paradigm

The system



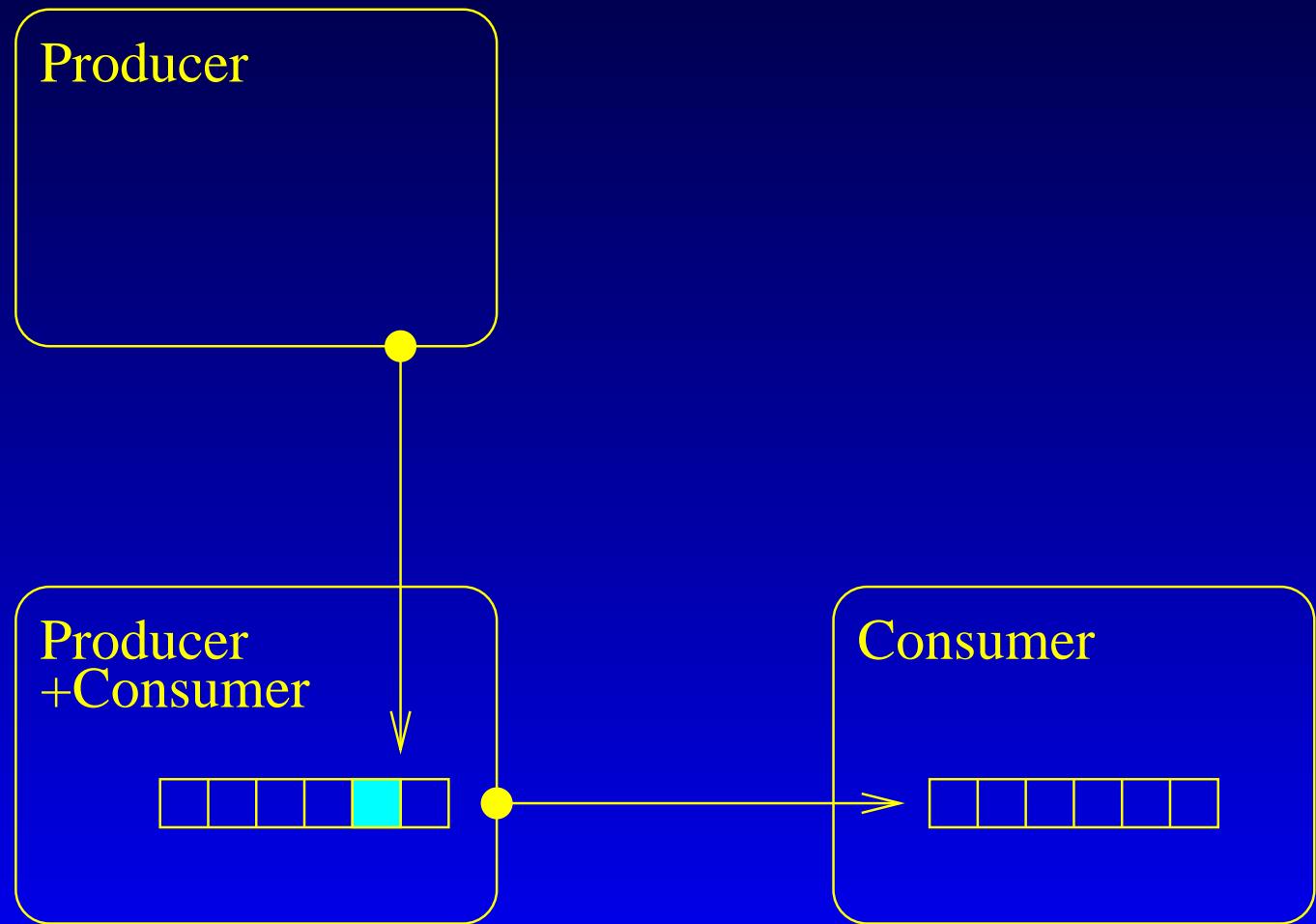
# Event Bus Paradigm

Production of an event



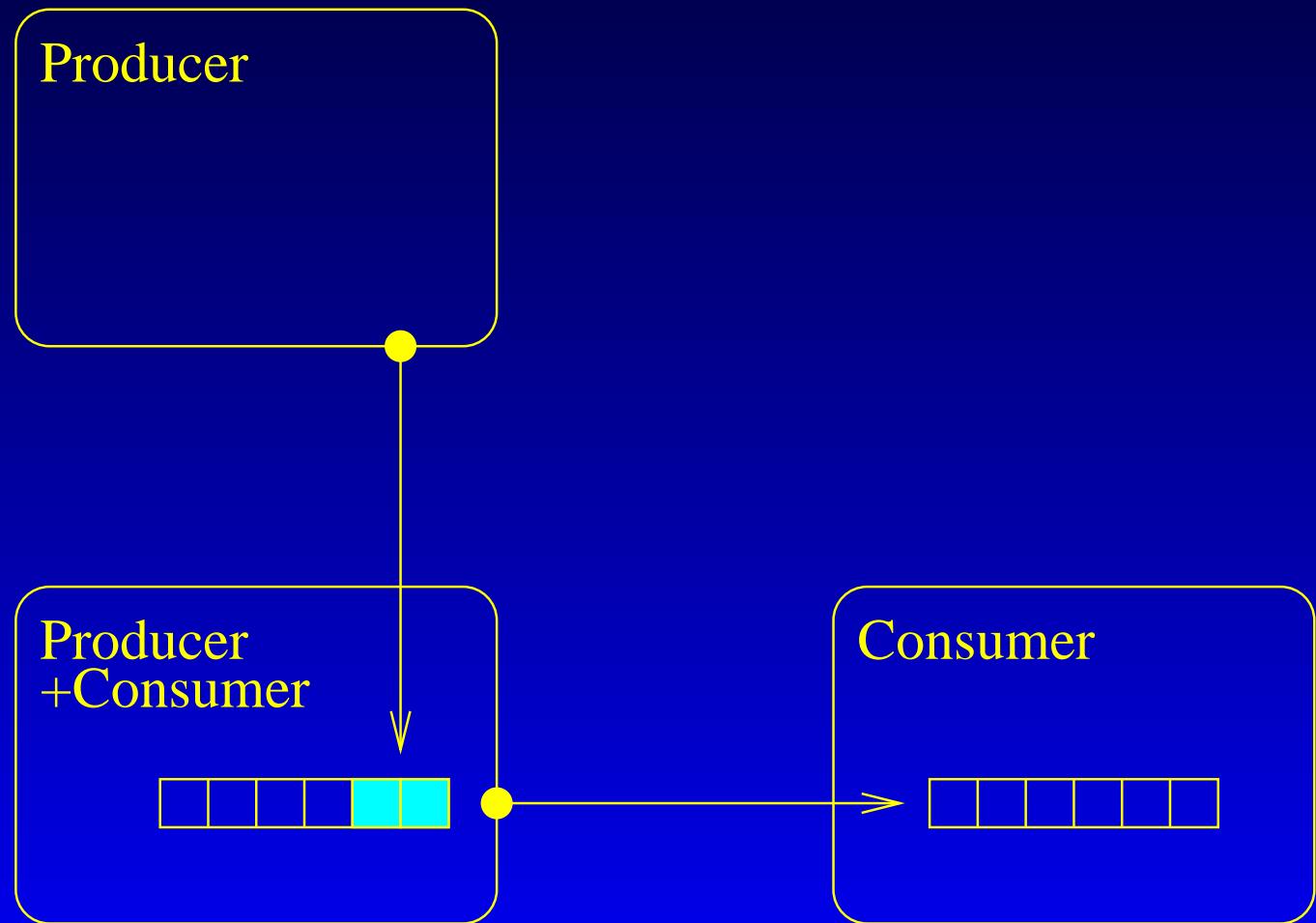
# Event Bus Paradigm

Production of another event (1)



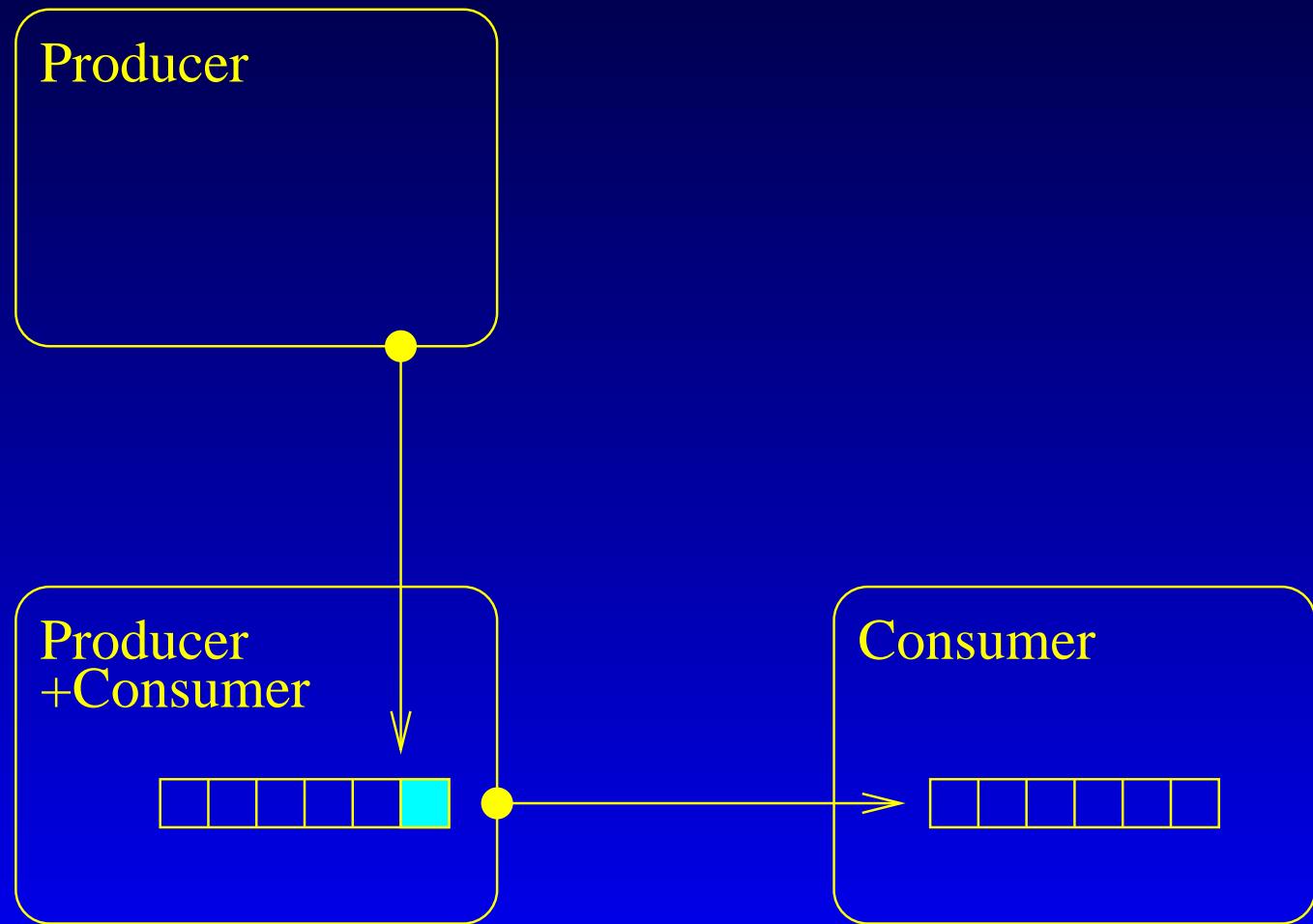
# Event Bus Paradigm

Production of another event (2)



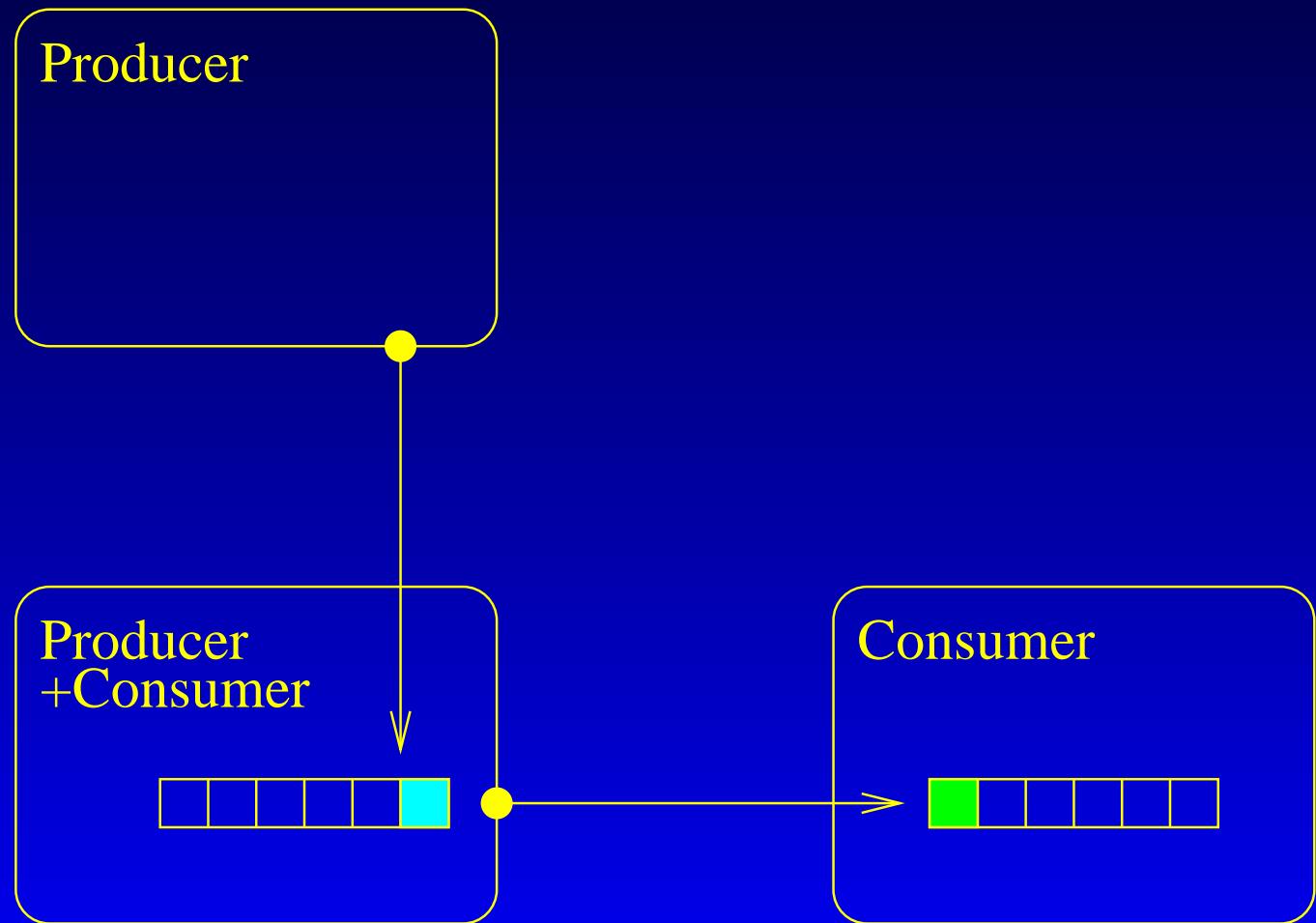
# Event Bus Paradigm

Consumption and treatment of the first event



# Event Bus Paradigm

This fires the production of an event



# Event Factories

# Event Factories

- Most of the time, events have short lifetime.

# Event Factories

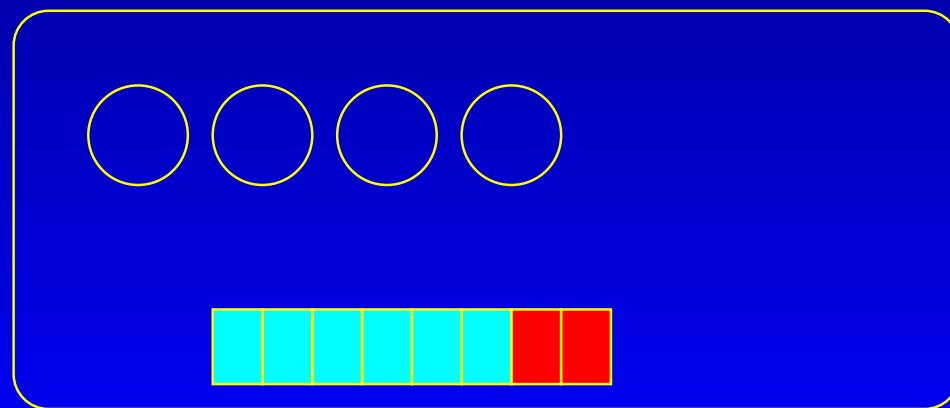
- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.

# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.

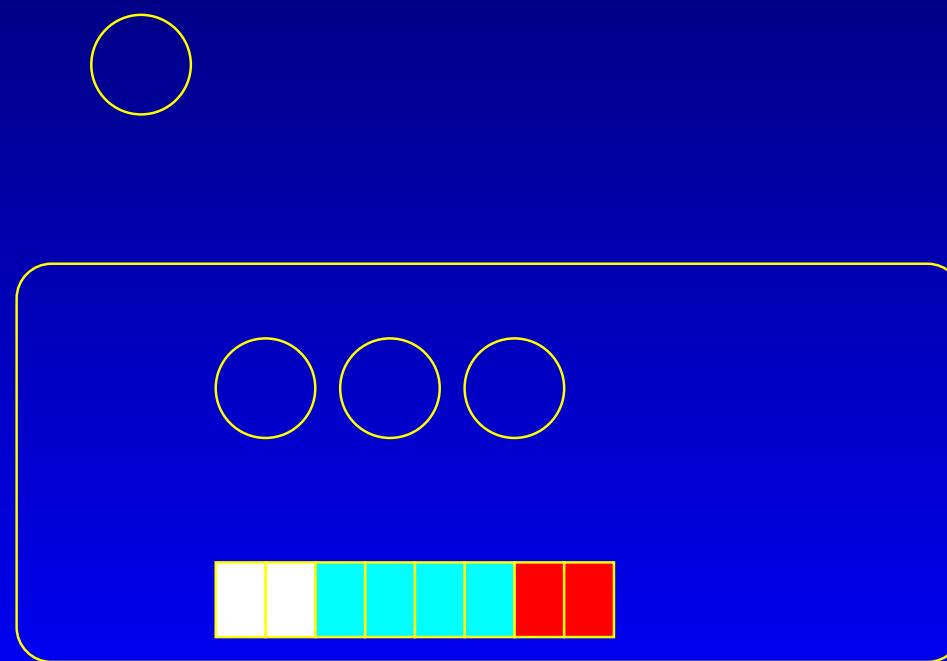
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



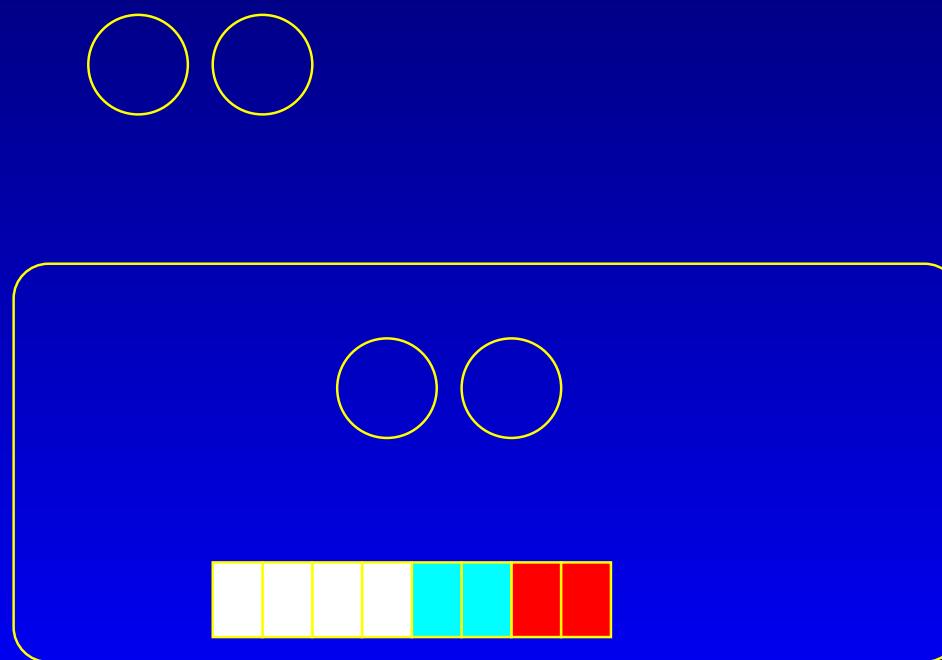
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



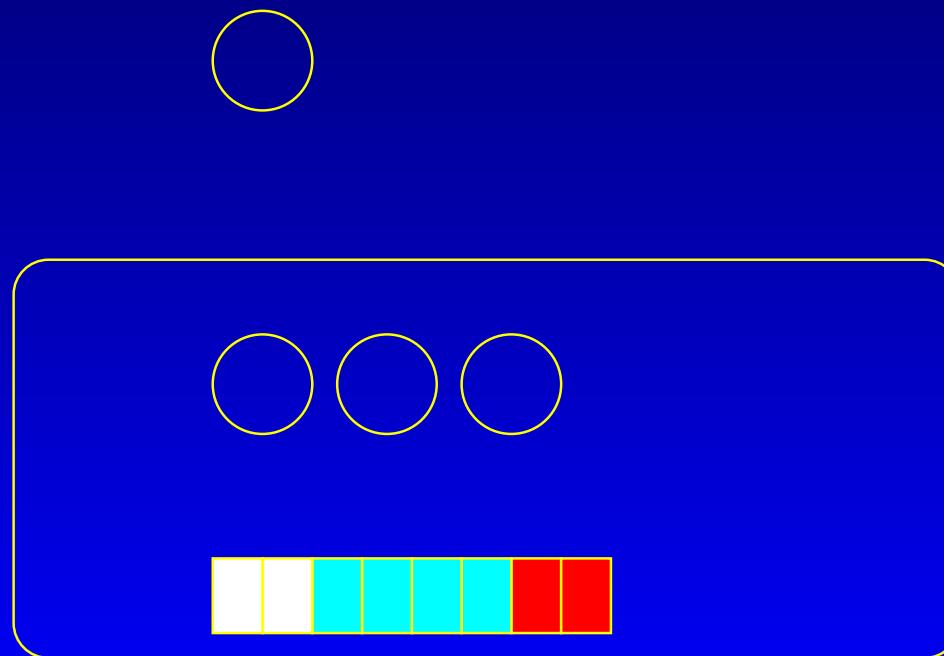
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



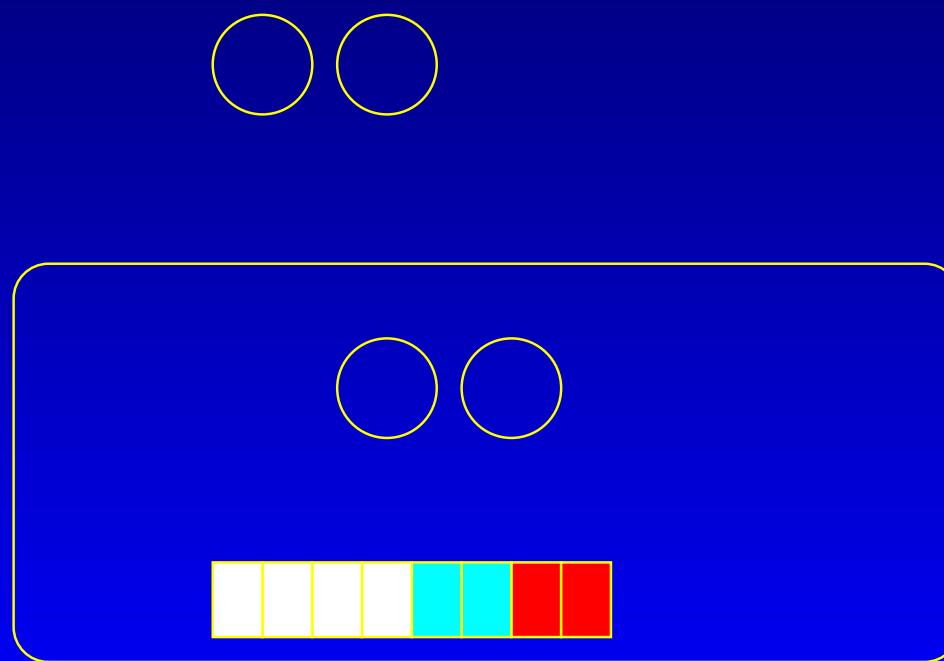
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



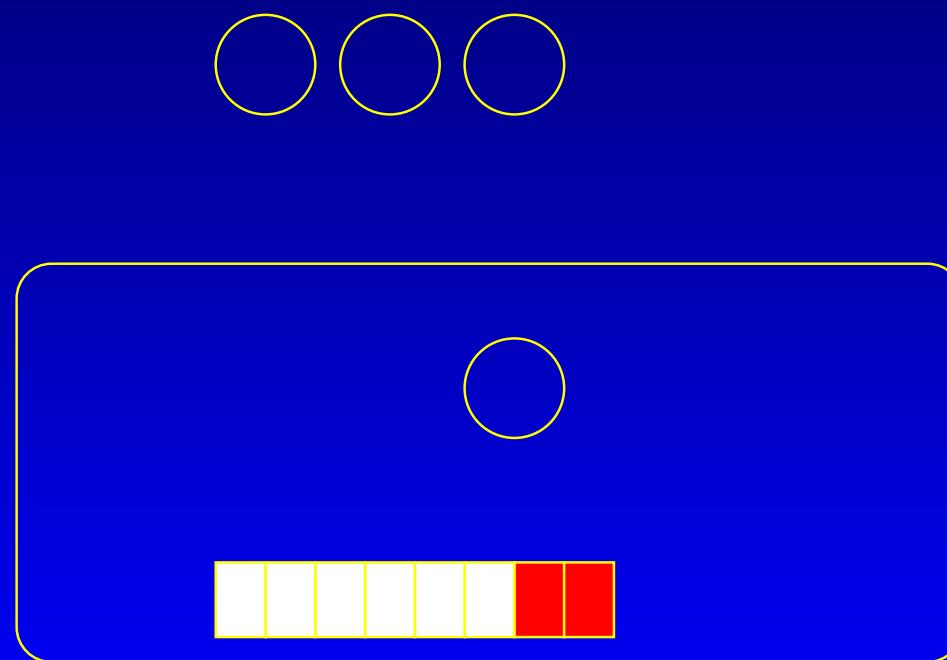
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



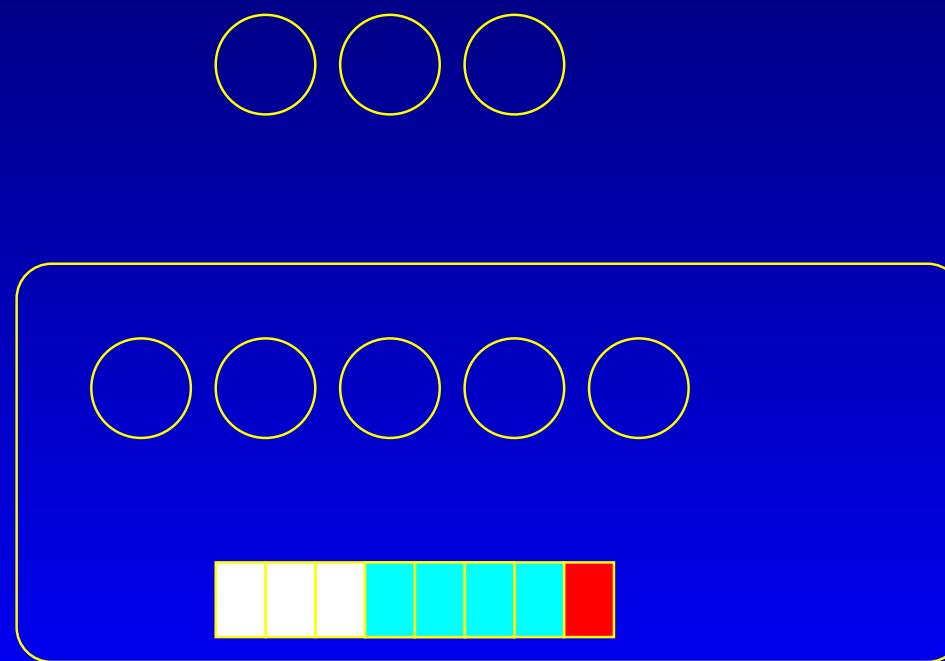
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



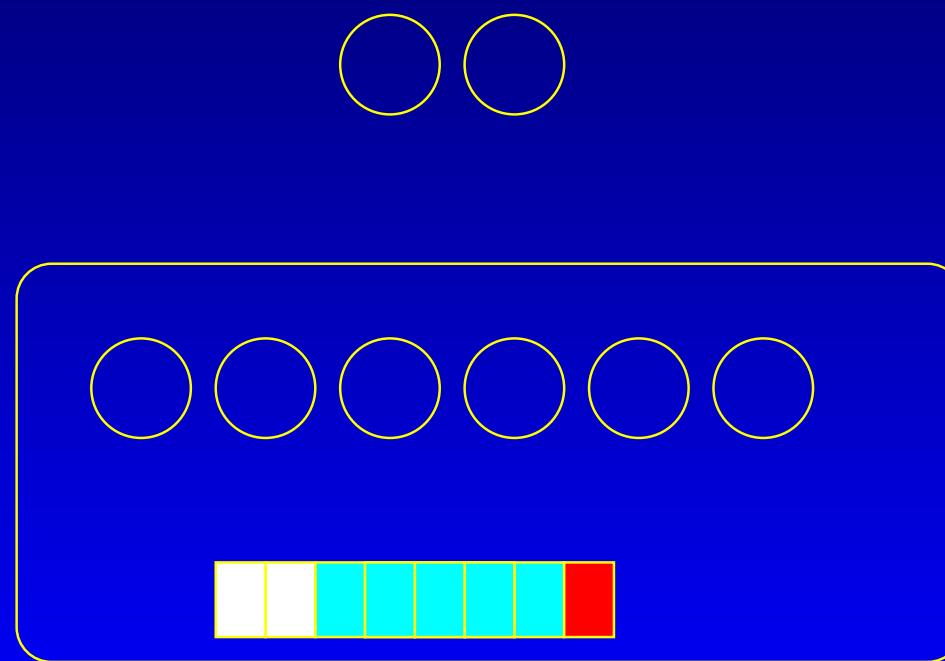
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



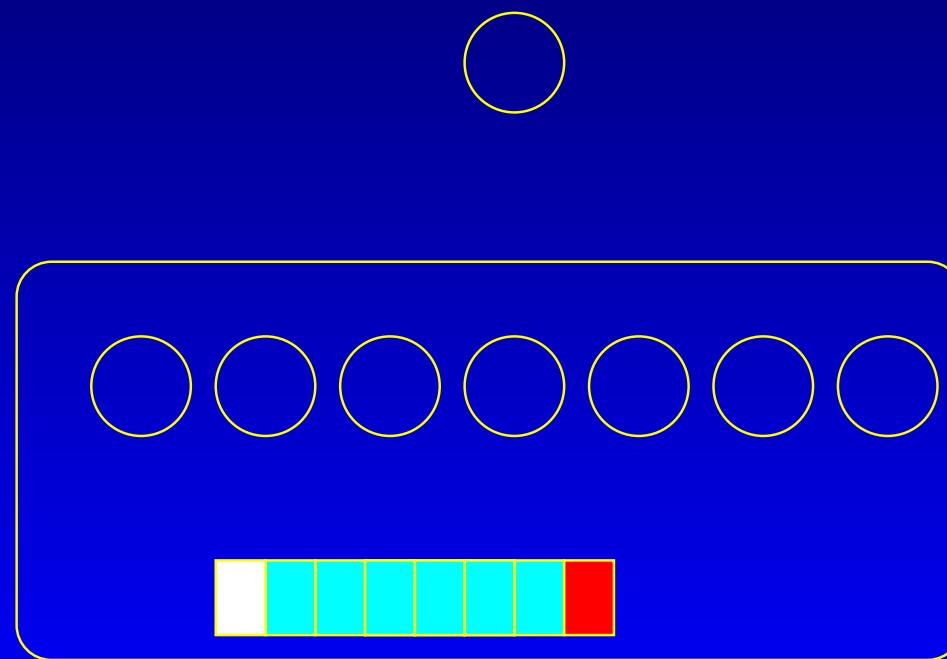
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



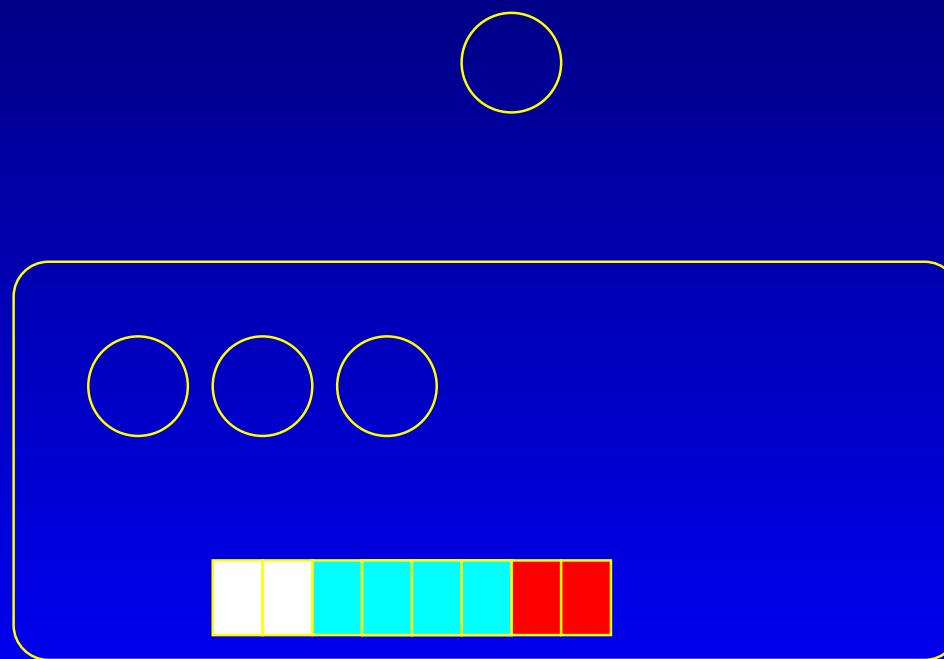
# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



# Event Factories

- Most of the time, events have short lifetime.
- We do not master the behavior of the internal GC of the JVM.
- It is better to rely on our own memory management system.



# Network Integration

# Network Integration

- The bus event architecture is basically local.

# Network Integration

- The bus event architecture is basically local.
- However, it needs to be extended by some mean to the network

# Network Integration

- The bus event architecture is basically local.
- However, it needs to be extended by some mean to the network
- To leave the programmer the choice of its network protocol, we have chosen to integrate the network by the mean of two specials consumer/producer :

# Network Integration

- The bus event architecture is basically local.
- However, it needs to be extended by some mean to the network
- To leave the programmer the choice of its network protocol, we have chosen to integrate the network by the mean of two specials consumer/producer :
  - A generic consumer of events that are to be sent over the network, called **notifier**.

# Network Integration

- The bus event architecture is basically local.
- However, it needs to be extended by some mean to the network
- To leave the programmer the choice of its network protocol, we have chosen to integrate the network by the mean of two specials consumer/producer :
  - A generic consumer of events that are to be sent over the network, called **notifier**.
  - A generic producer of events that are to be received from the network, called **listener**.

# Network Integration

# Network Integration

- There exists multiple instantiations of notifiers and their associated listeners :

# Network Integration

- There exists multiple instantiations of notifiers and their associated listeners :
  - UDP

# Network Integration

- There exists multiple instantiations of notifiers and their associated listeners :
  - UDP
  - UDP multicast

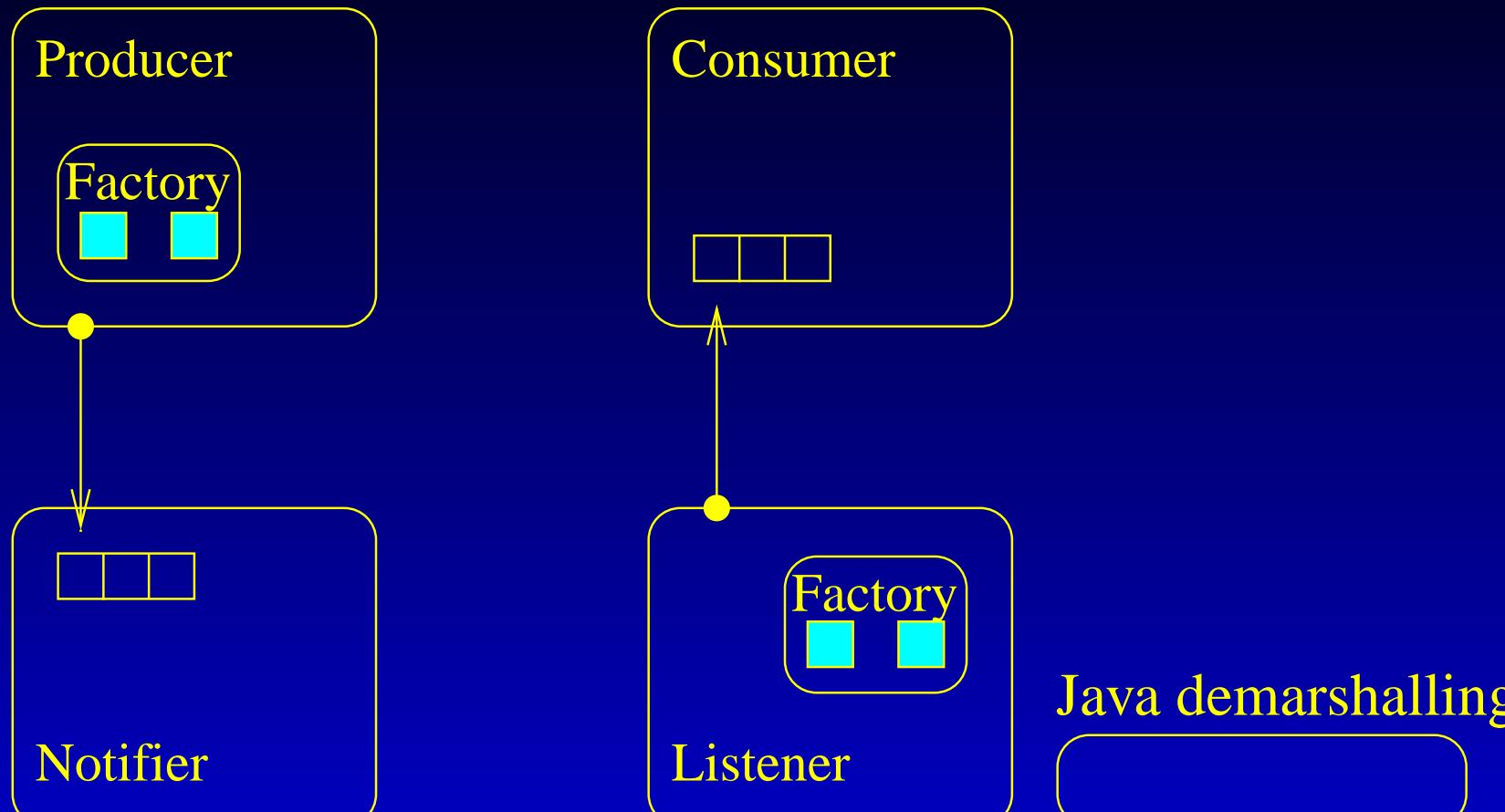
# Network Integration

- There exists multiple instantiations of notifiers and their associated listeners :
  - UDP
  - UDP multicast
  - TCP

# Network Integration

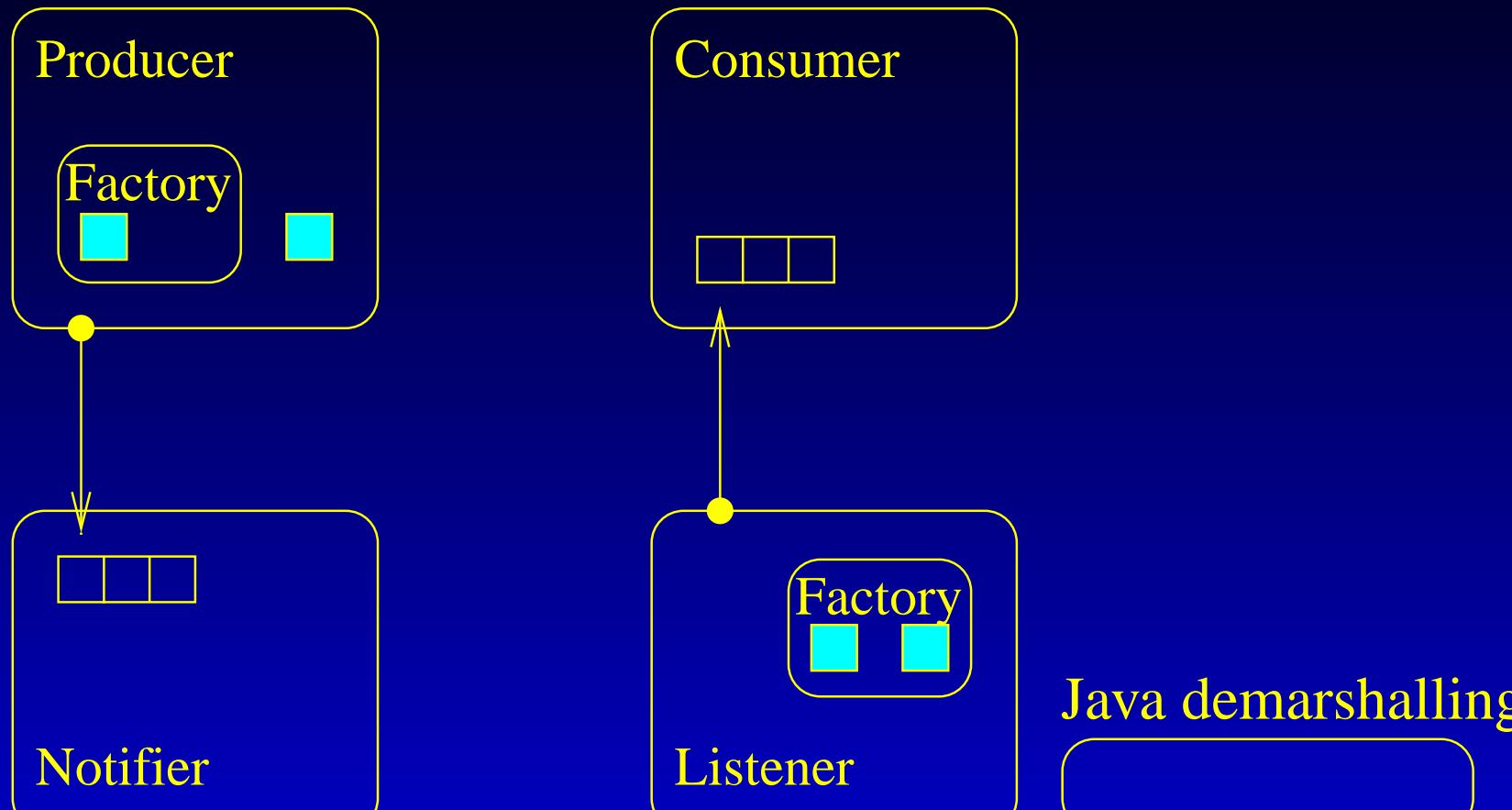
- There exists multiple instantiations of notifiers and their associated listeners :
  - UDP
  - UDP multicast
  - TCP
- The combination of factories and serialization/deserialization of events is problematic (because of Java)

# Network Integration



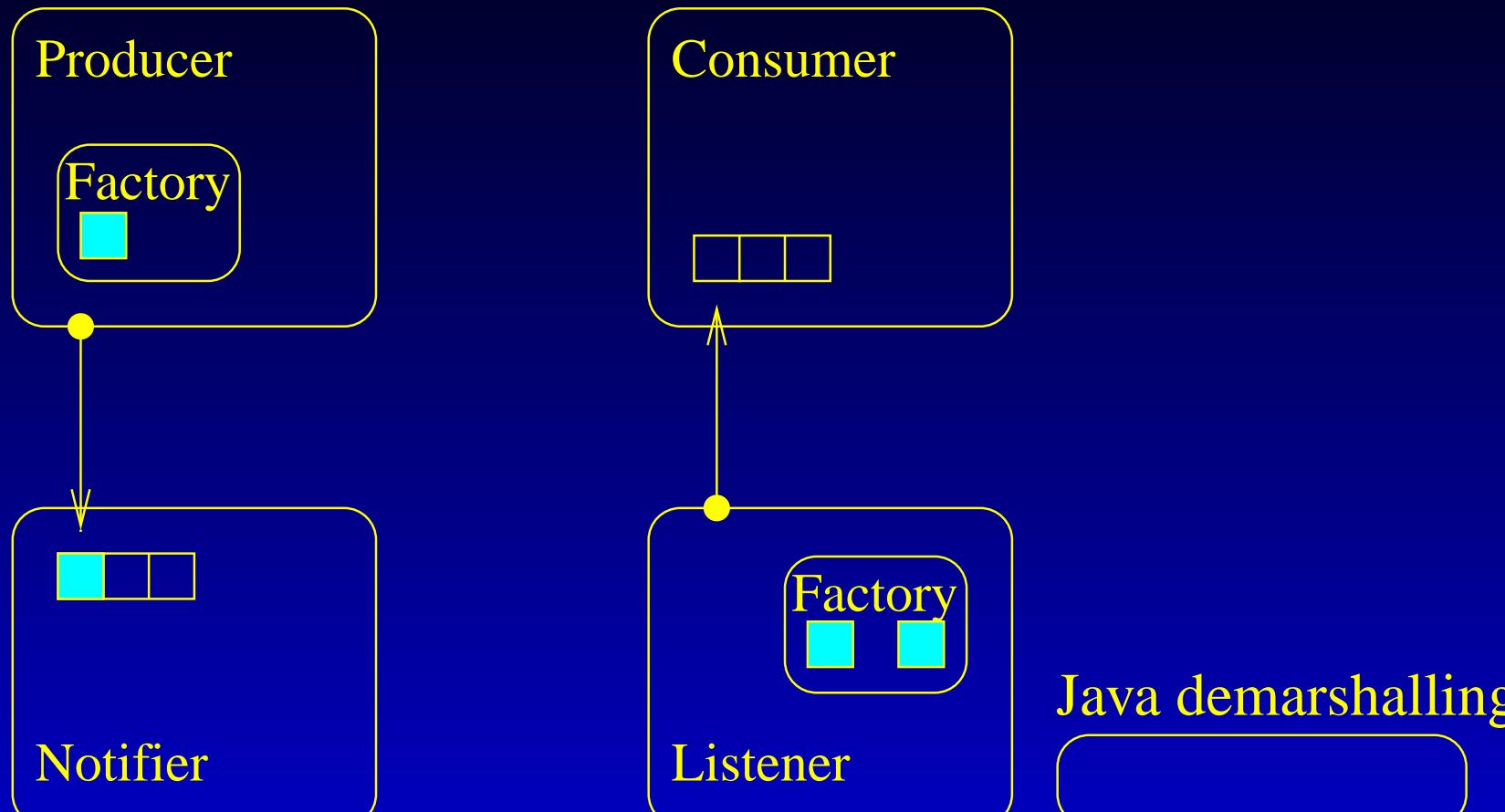
Network medium

# Network Integration



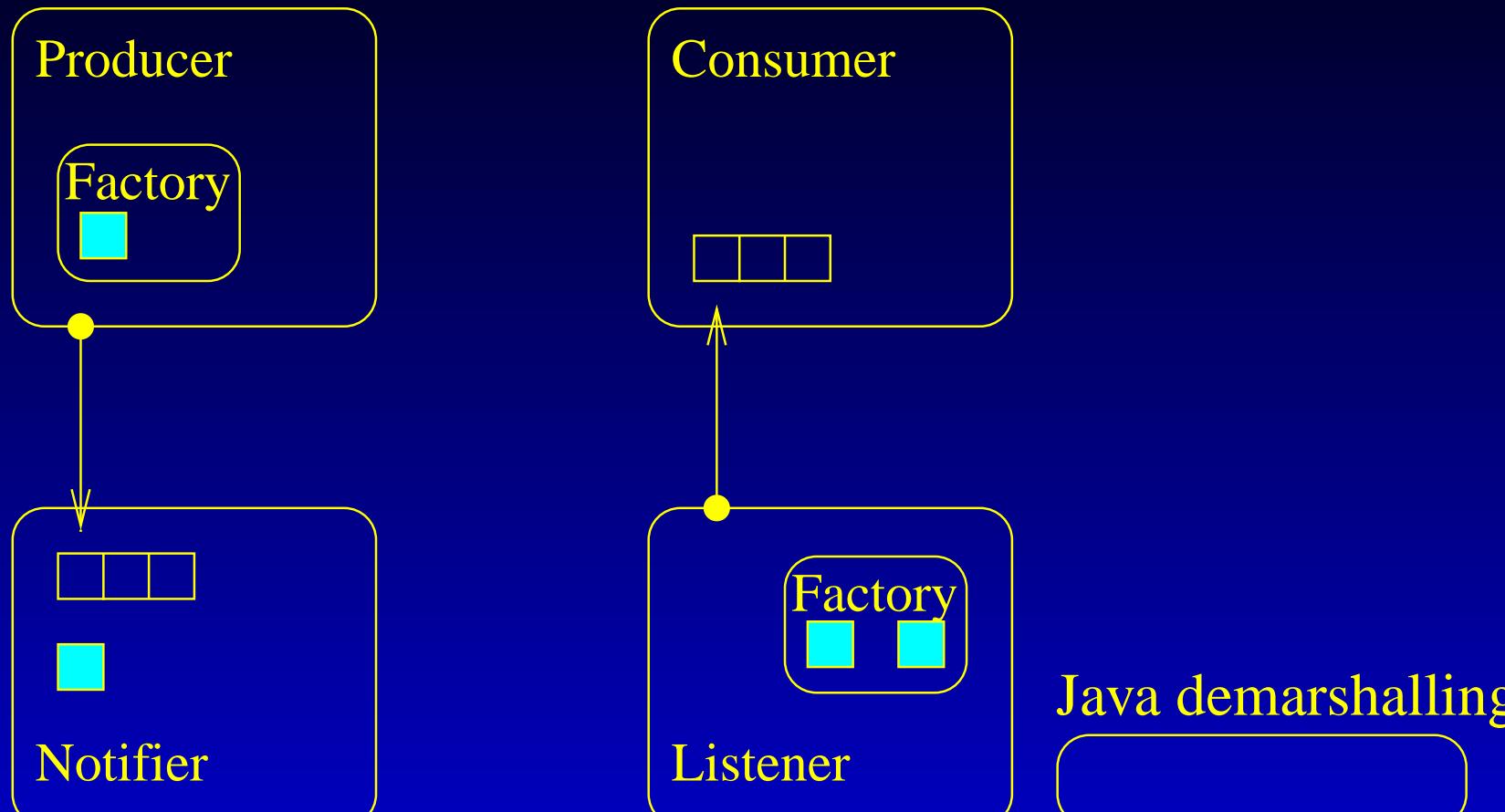
Network medium

# Network Integration



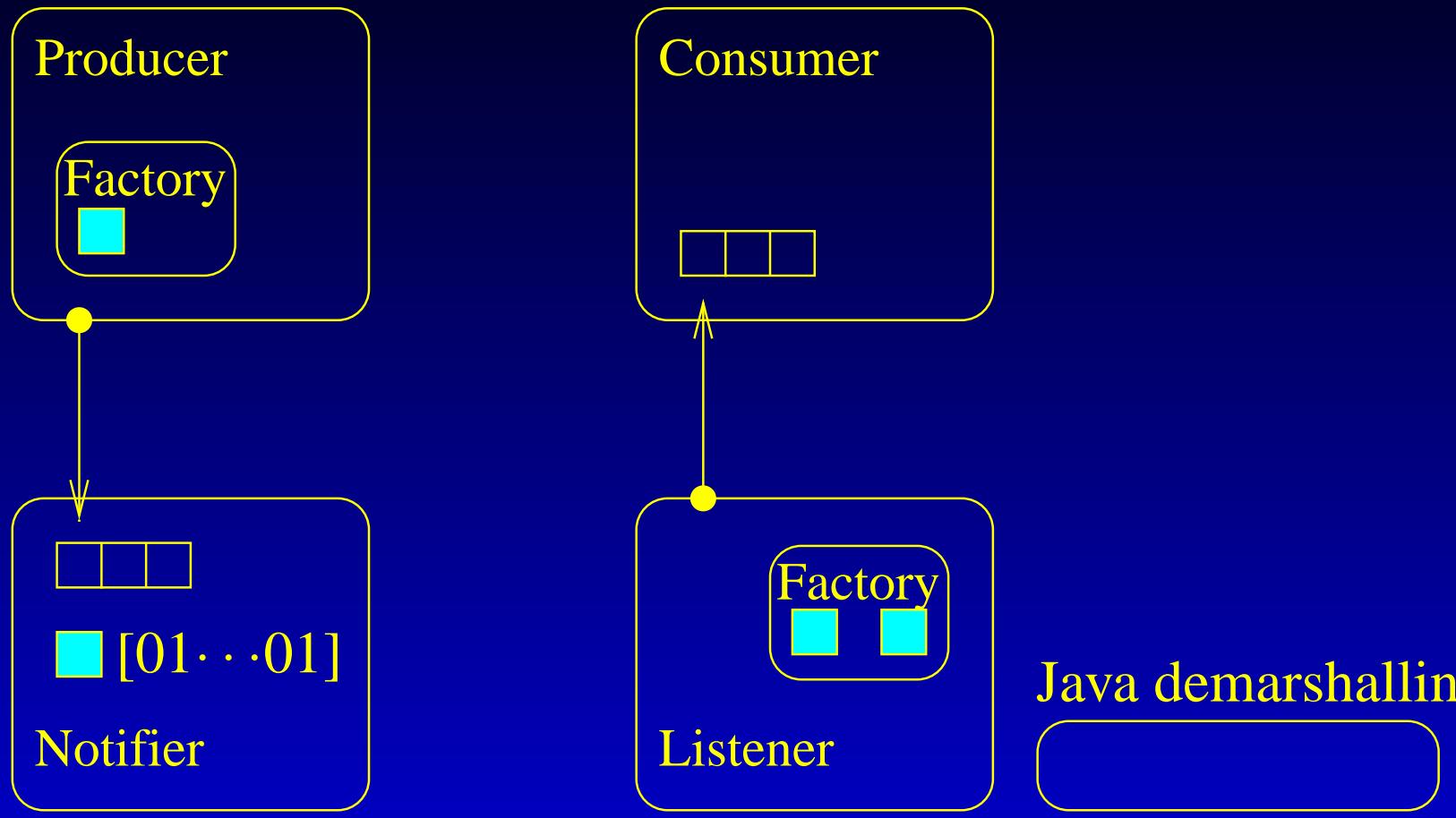
Network medium

# Network Integration



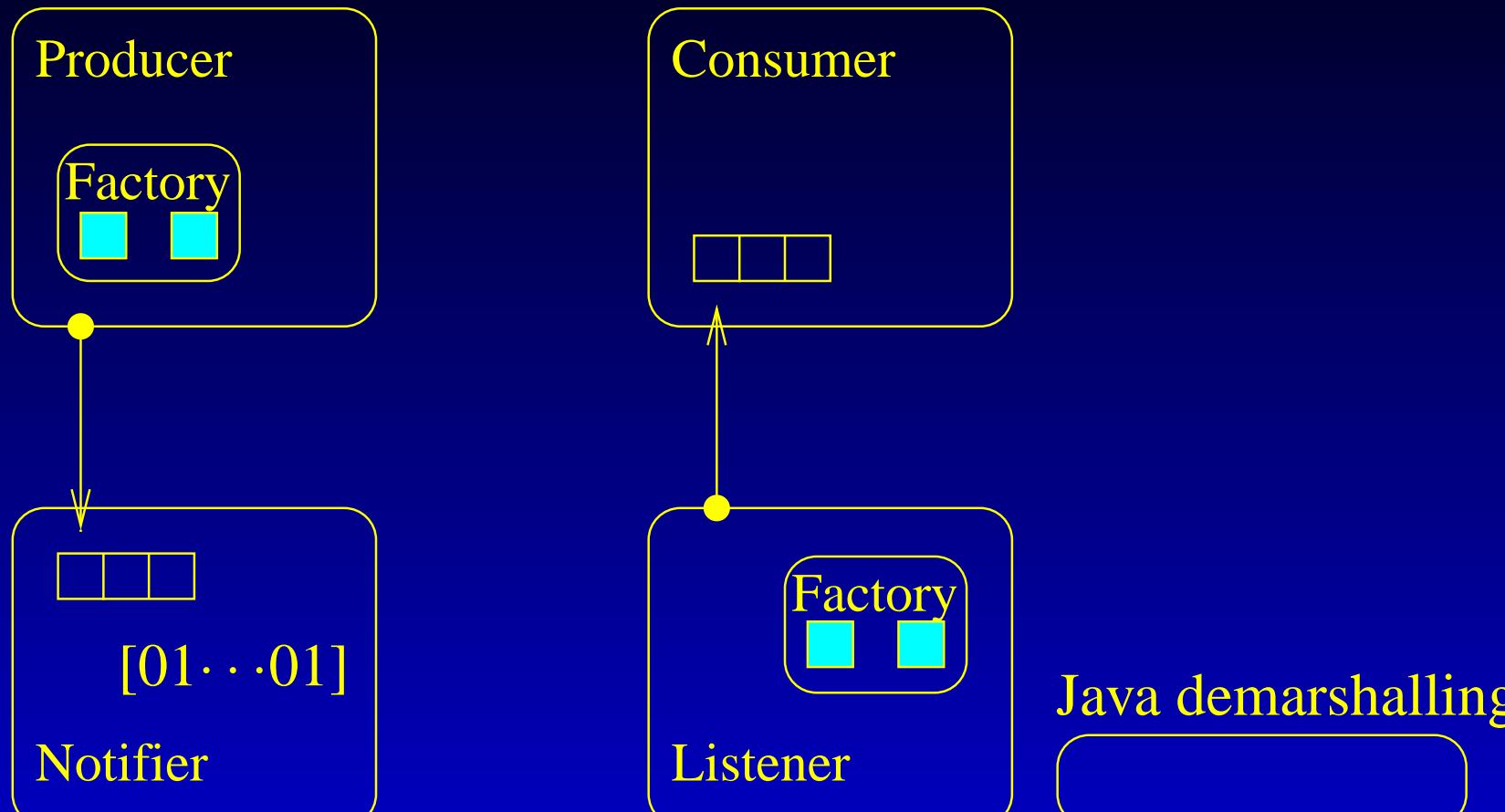
Network medium

# Network Integration

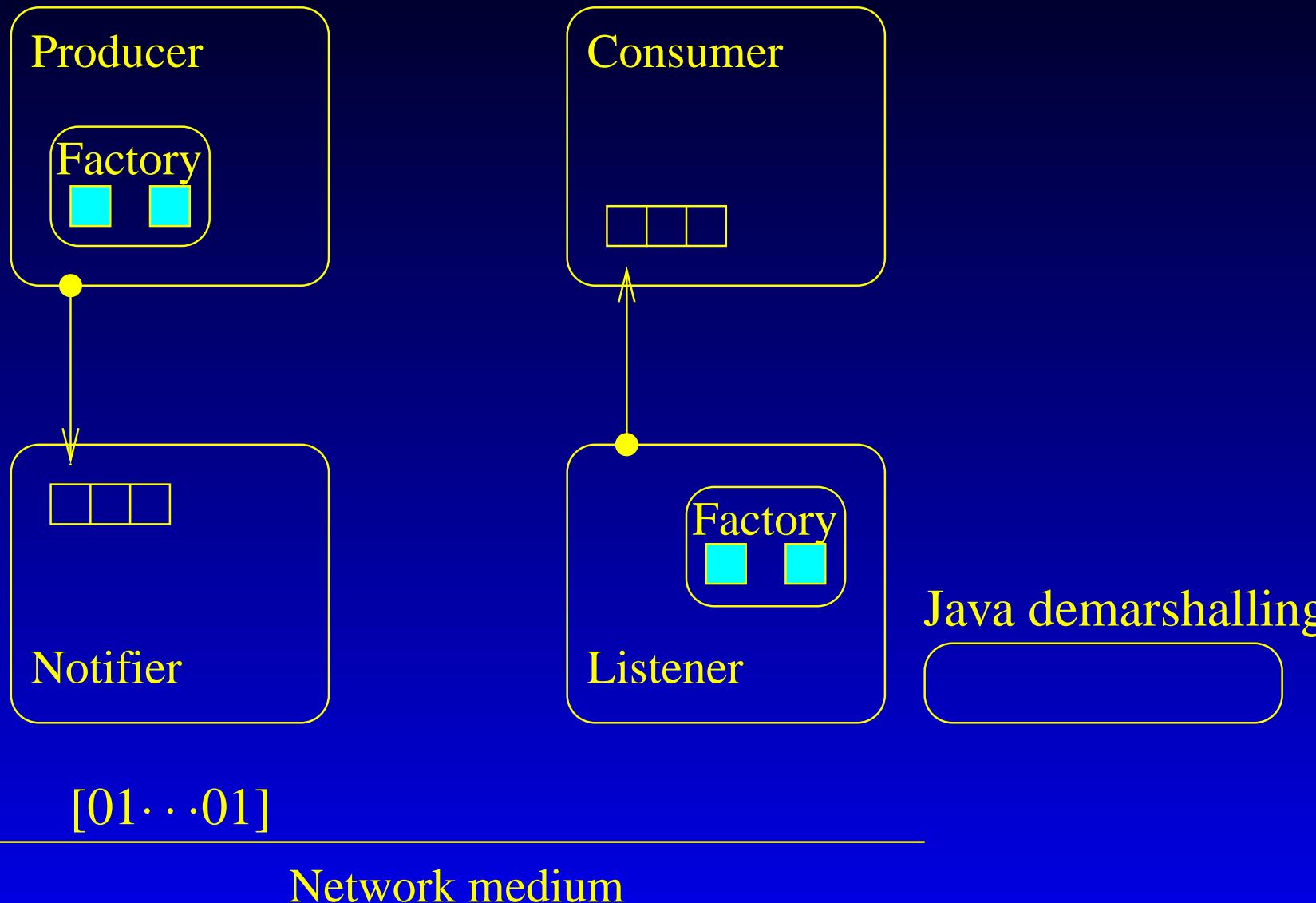


Network medium

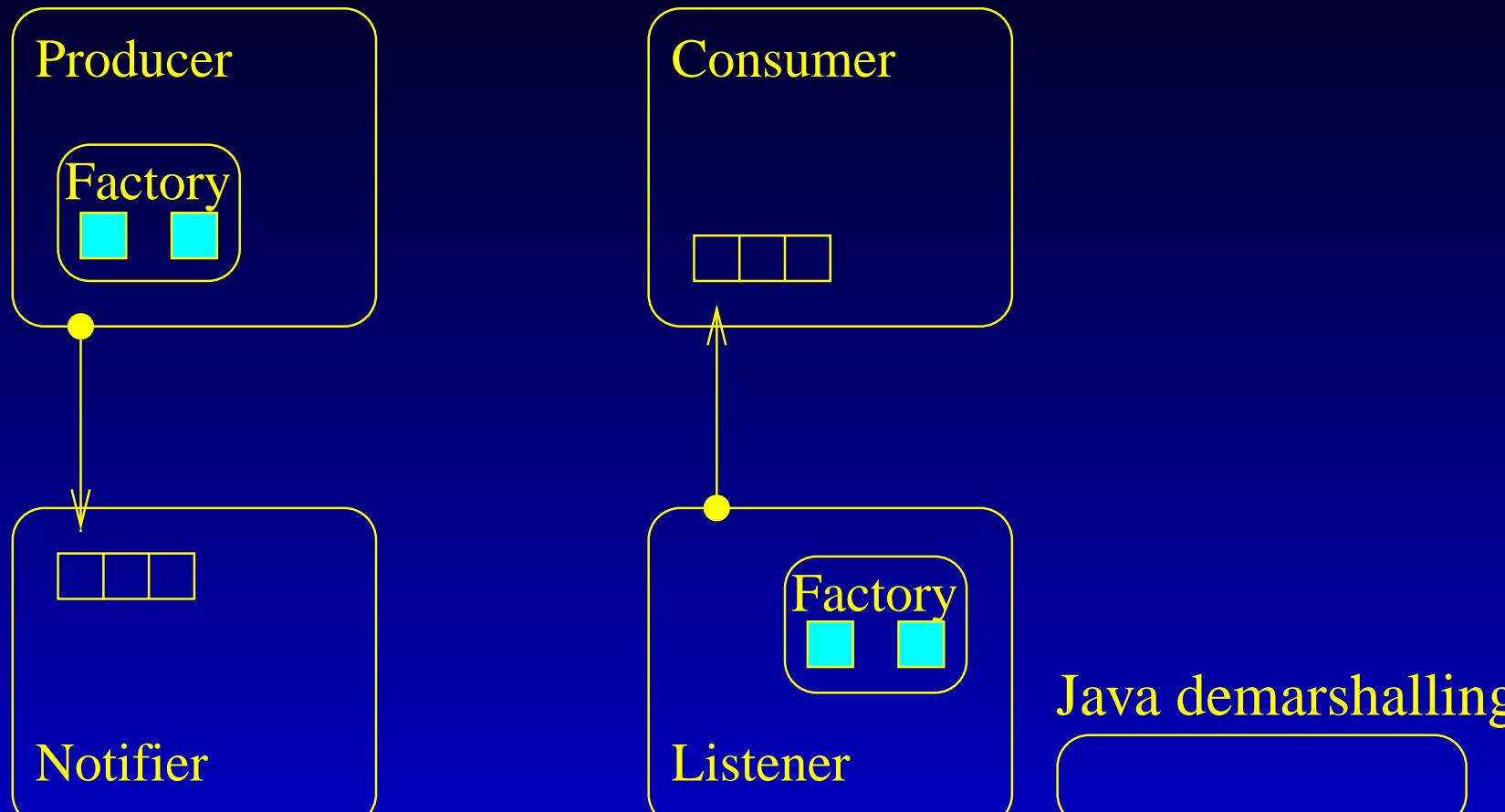
# Network Integration



# Network Integration



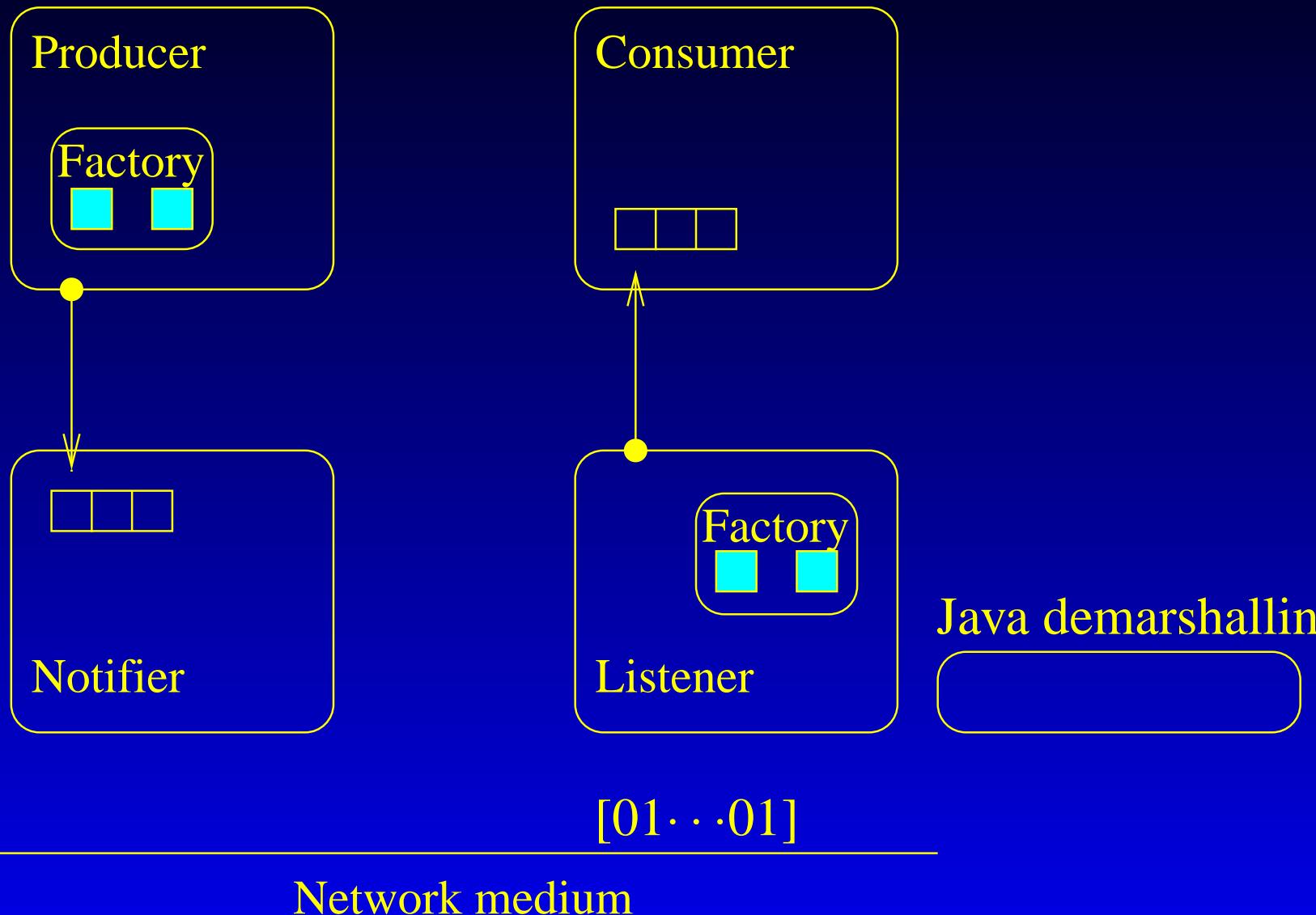
# Network Integration



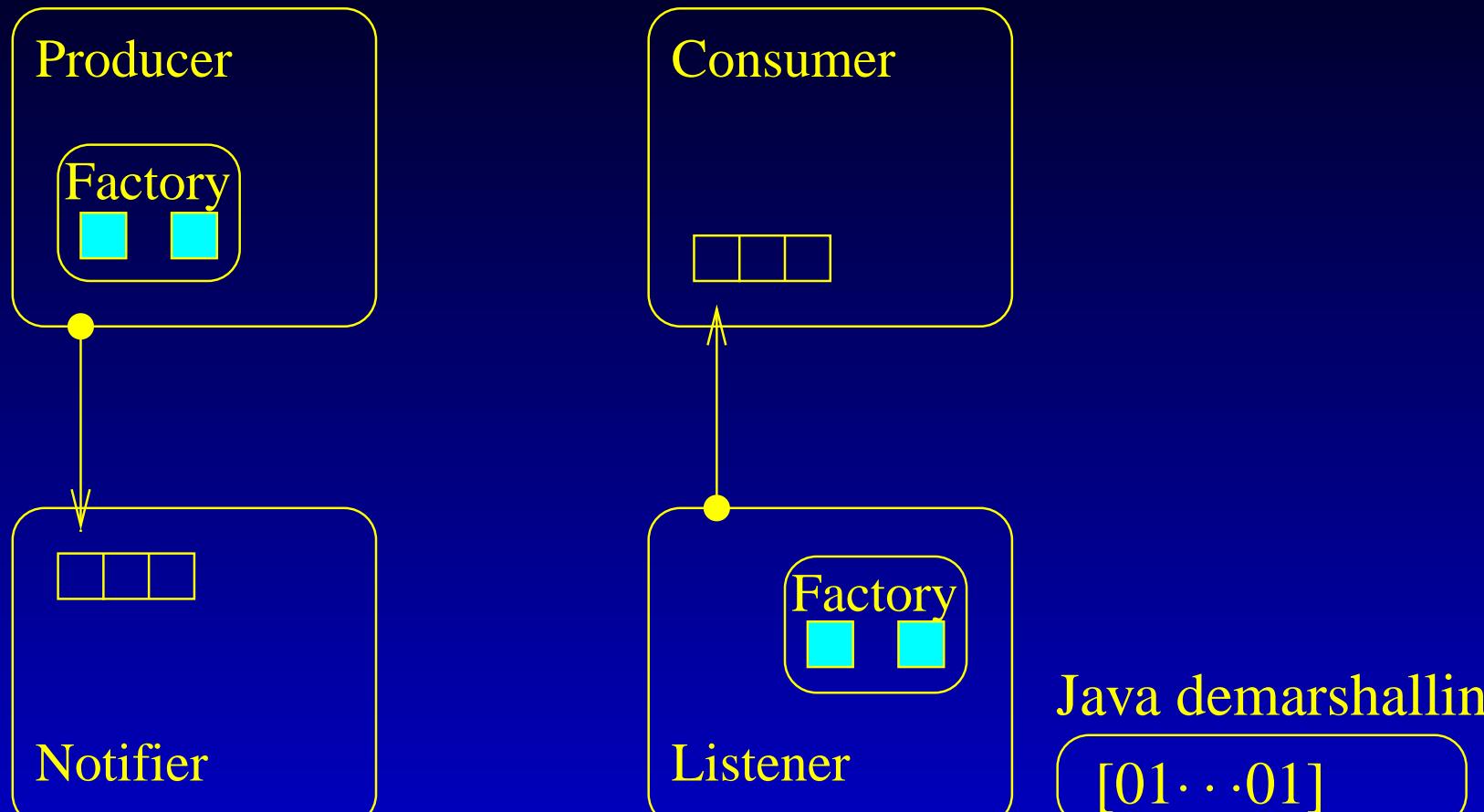
[01...01]

Network medium

# Network Integration

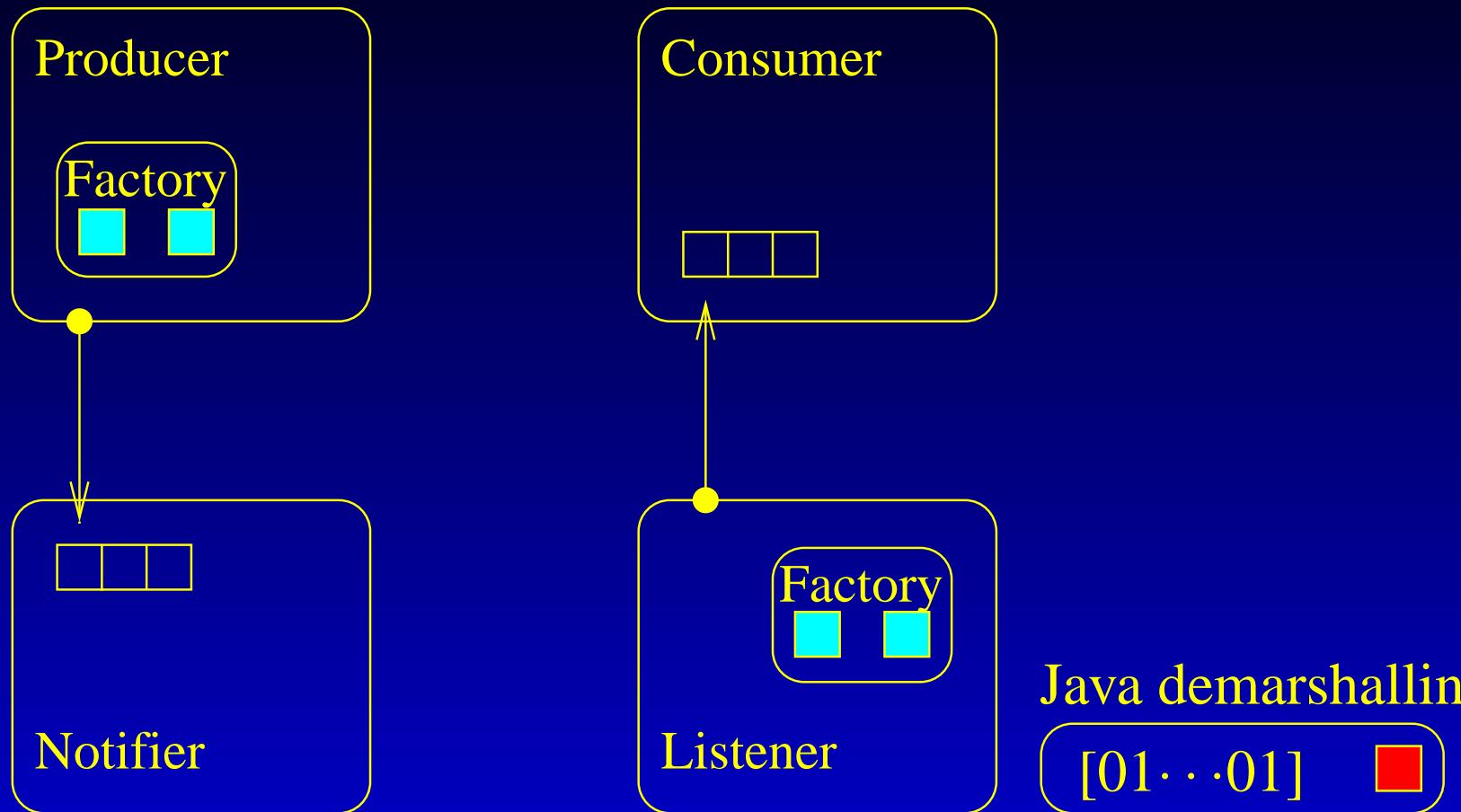


# Network Integration



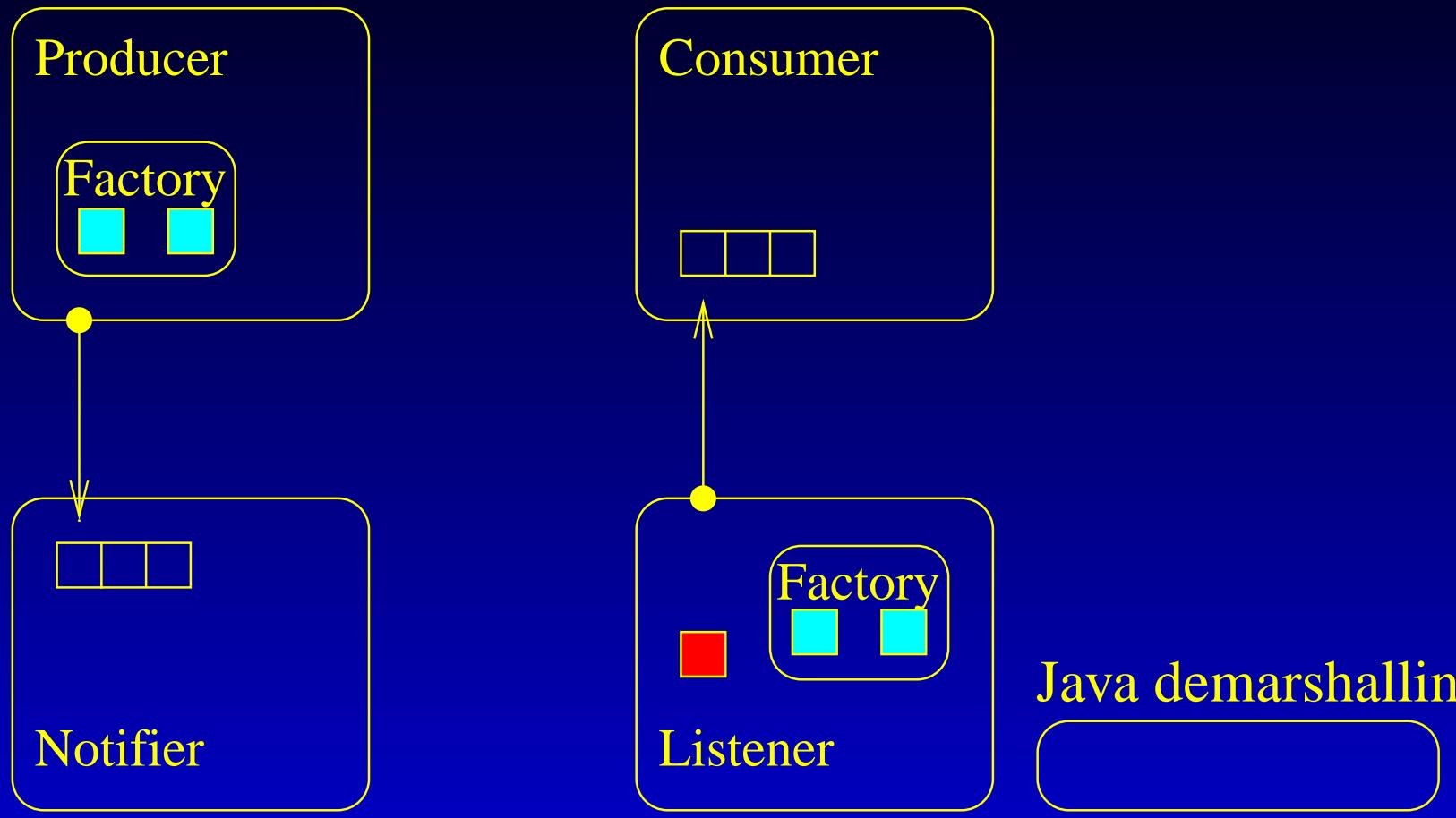
Network medium

# Network Integration



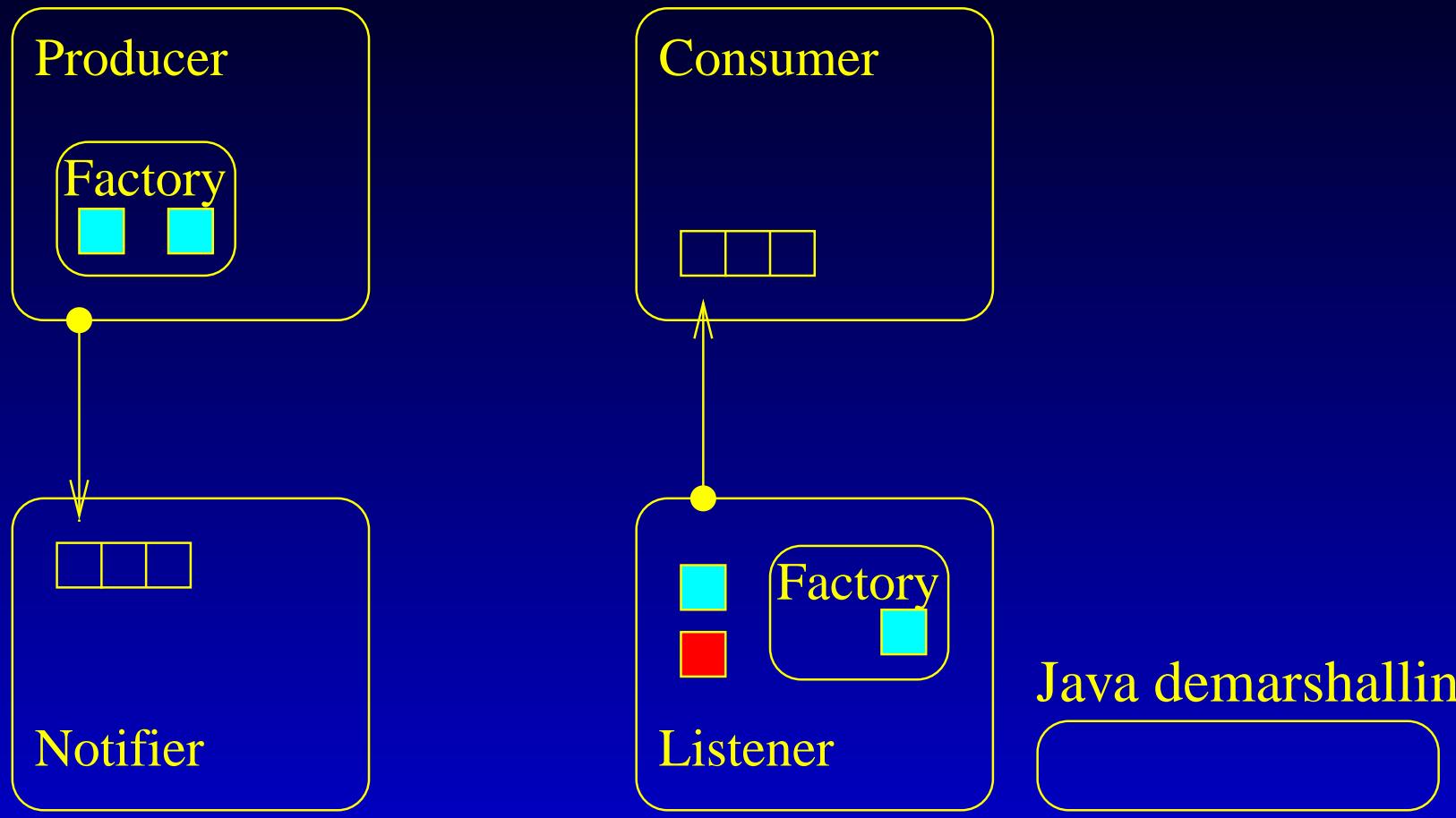
Network medium

# Network Integration



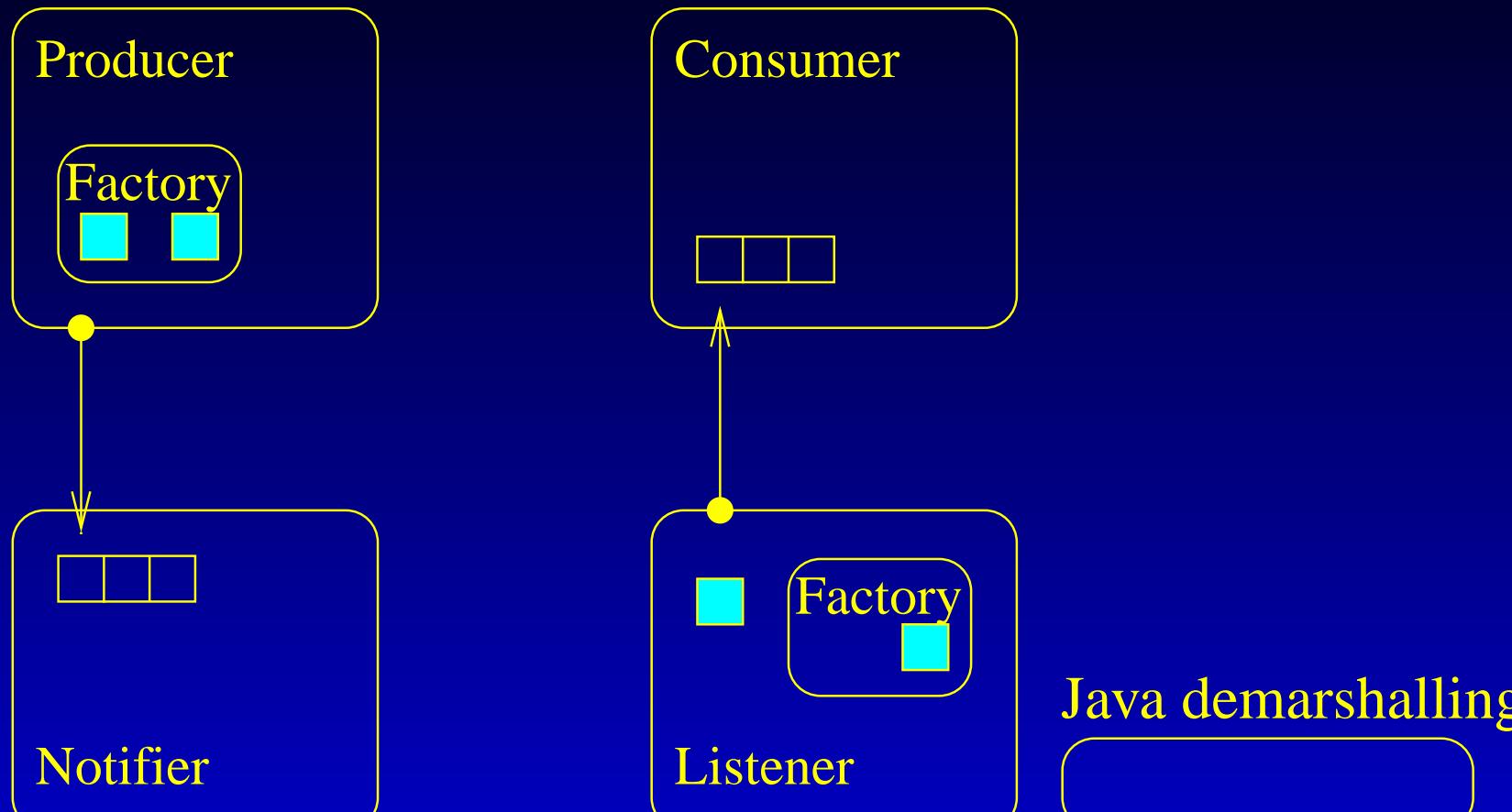
Network medium

# Network Integration



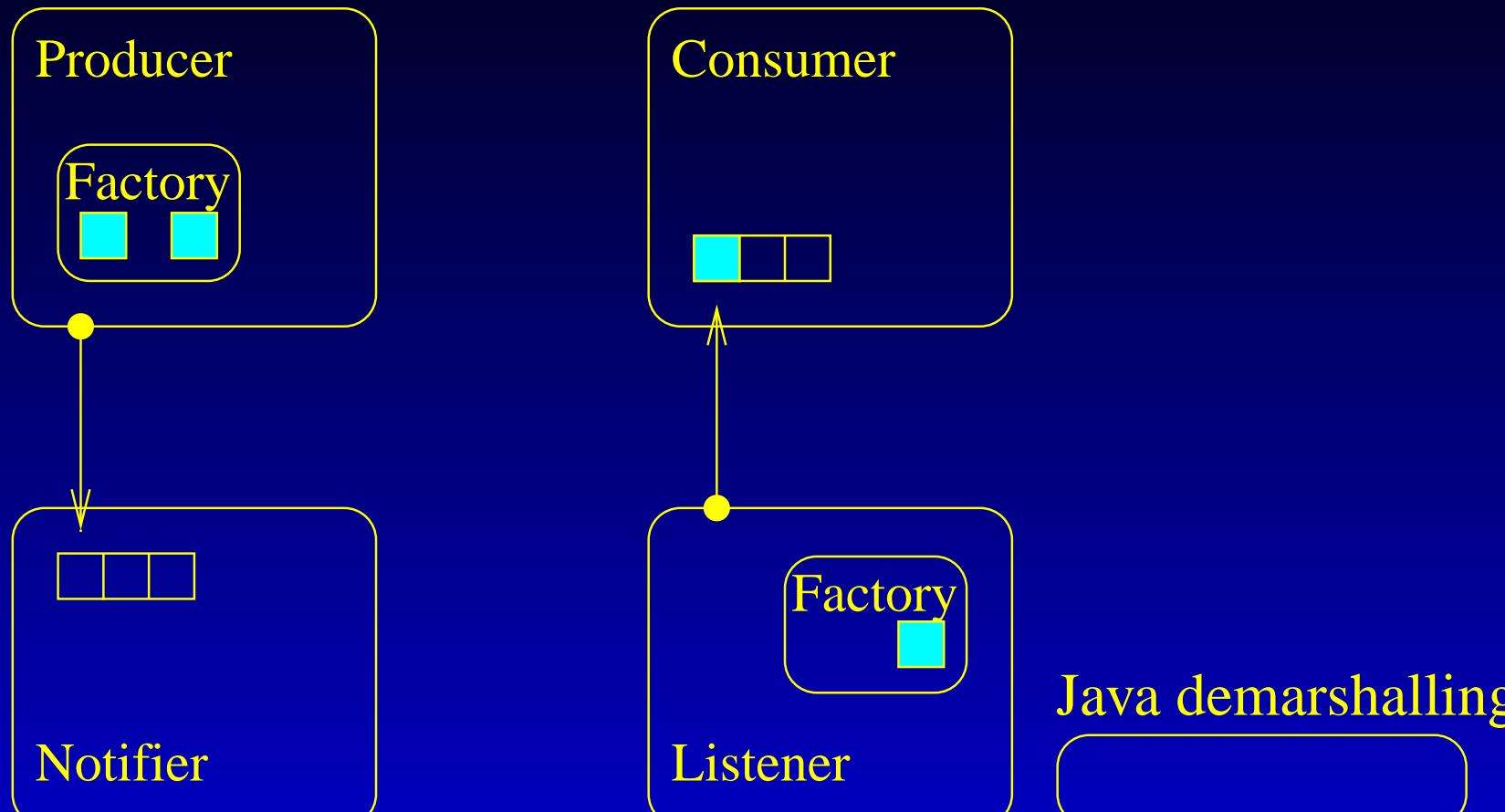
Network medium

# Network Integration



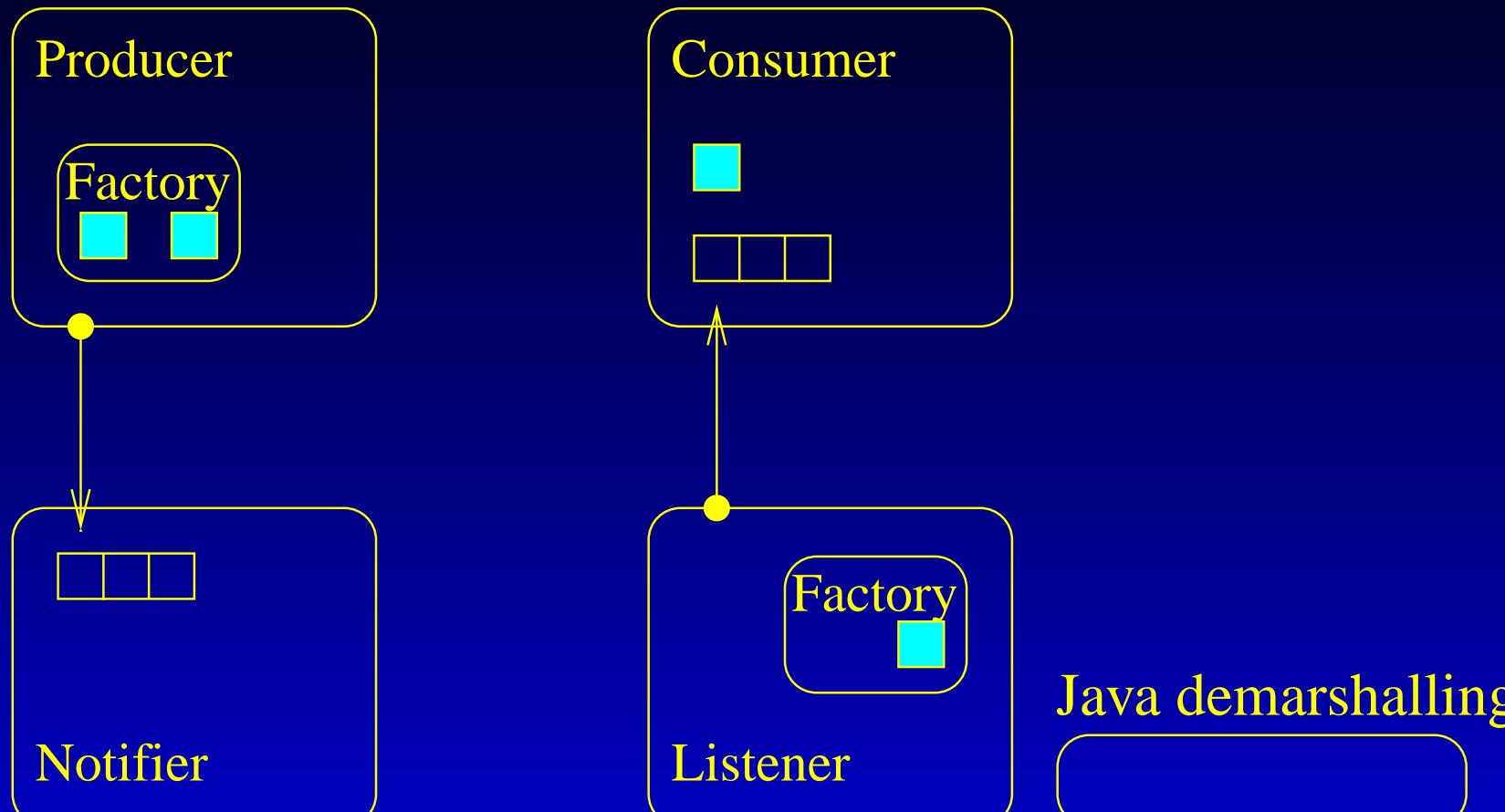
Network medium

# Network Integration



Network medium

# Network Integration



# Hierarchical Marshalling

# Hierarchical Marshalling

- Even though Eva is structuring the protocols as graphs, the traditional protocol stack often reappears.

# Hierarchical Marshalling

- Even though Eva is structuring the protocols as graphs, the traditional protocol stack often reappears.
- Indeed, it is very current that some events go through a chain of producers/consumers being further interpreted at each level :

# Hierarchical Marshalling

- Even though Eva is structuring the protocols as graphs, the traditional protocol stack often reappears.
- Indeed, it is very current that some events go through a chain of producers/consumers being further interpreted at each level :
  - It is then possible for the event to be discarded at any stage of the process, since it is too old.

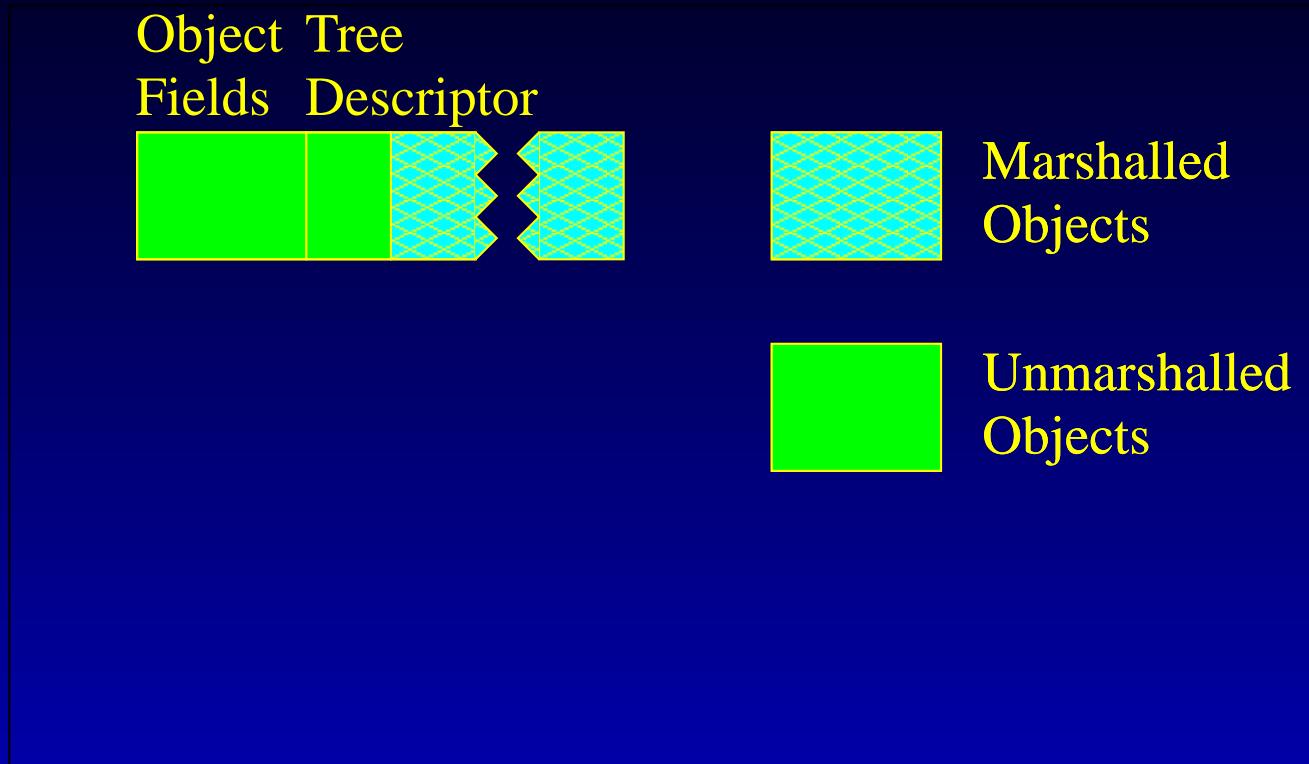
# Hierarchical Marshalling

- Even though Eva is structuring the protocols as graphs, the traditional protocol stack often reappears.
- Indeed, it is very current that some events go through a chain of producers/consumers being further interpreted at each level :
  - It is then possible for the event to be discarded at any stage of the process, since it is too old.
  - Or for a part of the event to be stored, while waiting for missing informations.

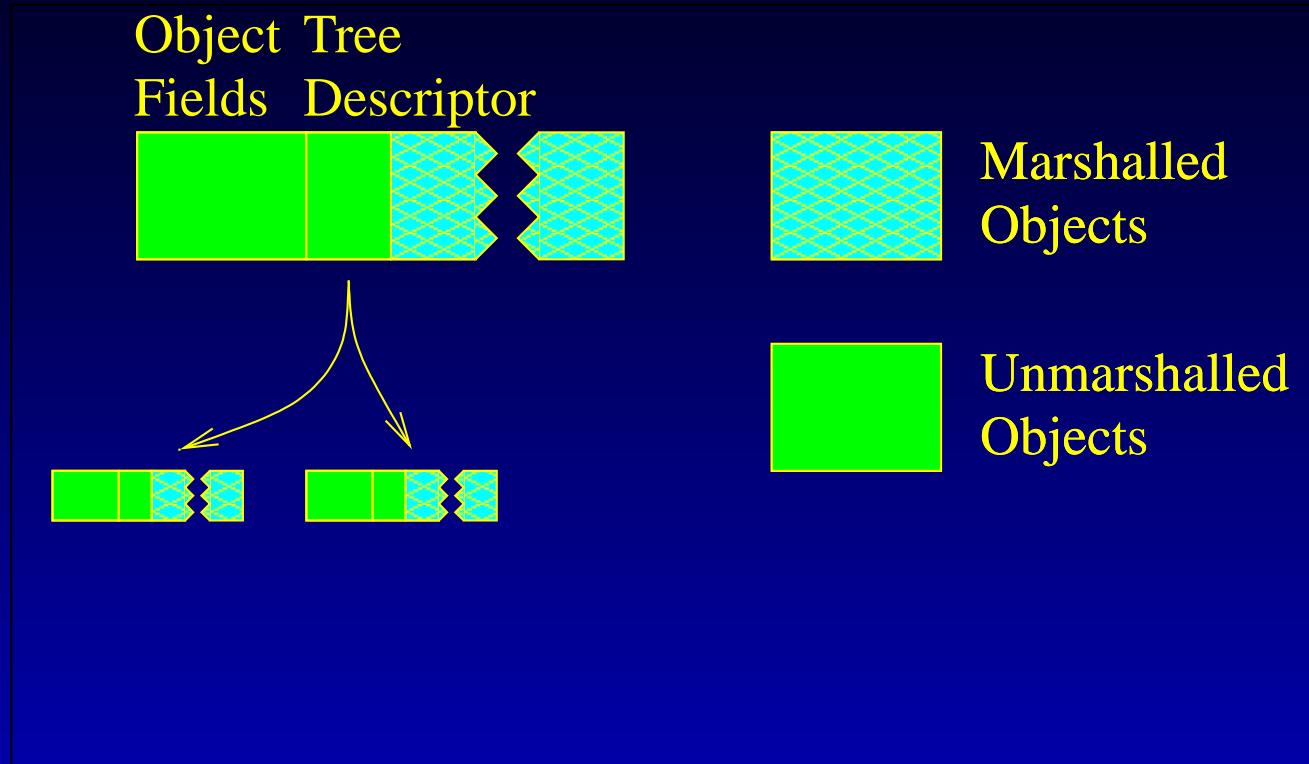
# Hierarchical Marshalling

- Even though Eva is structuring the protocols as graphs, the traditional protocol stack often reappears.
- Indeed, it is very current that some events go through a chain of producers/consumers being further interpreted at each level :
  - It is then possible for the event to be discarded at any stage of the process, since it is too old.
  - Or for a part of the event to be stored, while waiting for missing informations.
- This pleads in favor of a hierarchical structure for events.

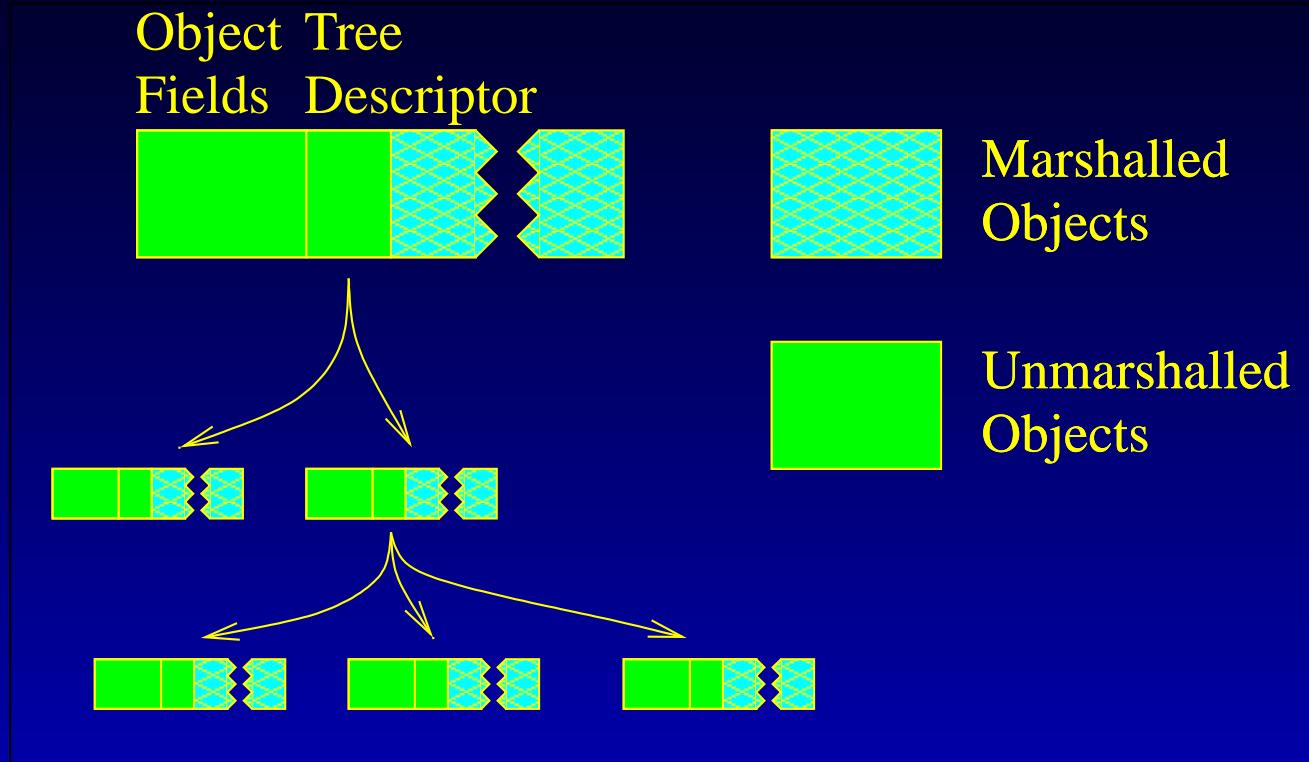
# Hierarchical Marshalling



# Hierarchical Marshalling



# Hierarchical Marshalling



# Hierarchical Marshalling

# Hierarchical Marshalling

- Events that can be sent over the network are provided with an interface to manage :

# Hierarchical Marshalling

- Events that can be sent over the network are provided with an interface to manage :

# Hierarchical Marshalling

- Events that can be sent over the network are provided with an interface to manage :
  - The tree structure of associated events

# Hierarchical Marshalling

- Events that can be sent over the network are provided with an interface to manage :
  - The tree structure of associated events
  - The serialisation/deserialisation of a topmost event (root of a tree)

# Conclusion about Eva

# Conclusion about Eva

- Eva is an event based architecture for building high level protocols

# Conclusion about Eva

- Eva is an event based architecture for building high level protocols
- It is efficient, while at the same time maintaining ease of use.

# Conclusion about Eva

- Eva is an event based architecture for building high level protocols
- It is efficient, while at the same time maintaining ease of use.
- It has been successfully used and intensively debugged during the coding of Adam.

# Adam an Agreement Framework for Building GCS

# Group Communication Services

# Group Communication Services

- Group Communication Services (GCS) have been developed to simplify the creation of distributed applications

# Group Communication Services

- Group Communication Services (GCS) have been developed to simplify the creation of distributed applications
- They provide strong semantics primitives :

# Group Communication Services

- Group Communication Services (GCS) have been developed to simplify the creation of distributed applications
- They provide strong semantics primitives :
  - Atomic Broadcast to provide a total order on the set of messages broadcast within the group

# Group Communication Services

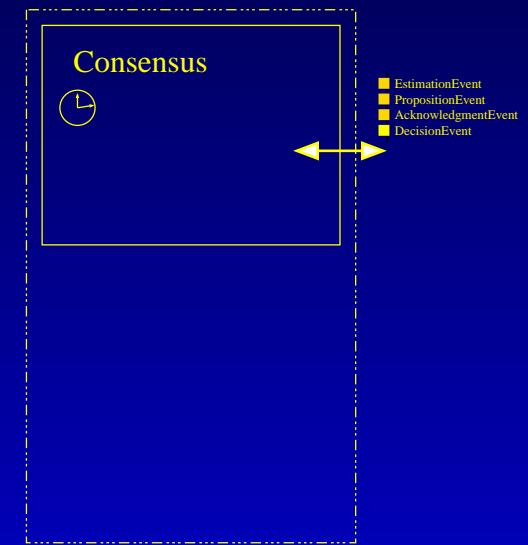
- Group Communication Services (GCS) have been developed to simplify the creation of distributed applications
- They provide strong semantics primitives :
  - Atomic Broadcast to provide a total order on the set of messages broadcast within the group
  - Membership to track the change of membership within the group

# Group Communication Services

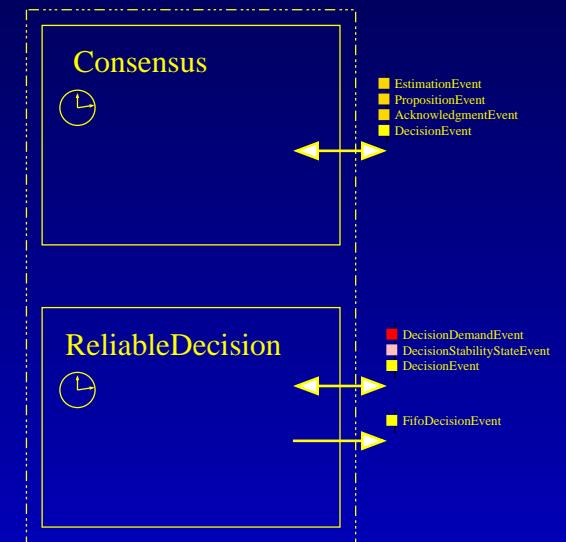
- Group Communication Services (GCS) have been developed to simplify the creation of distributed applications
- They provide strong semantics primitives :
  - Atomic Broadcast to provide a total order on the set of messages broadcast within the group
  - Membership to track the change of membership within the group
  - View Synchrony (VS) that aims at synchronizing the two previous services, by determining the set of messages to be ordered before a view change can take place

# Overview of Adam

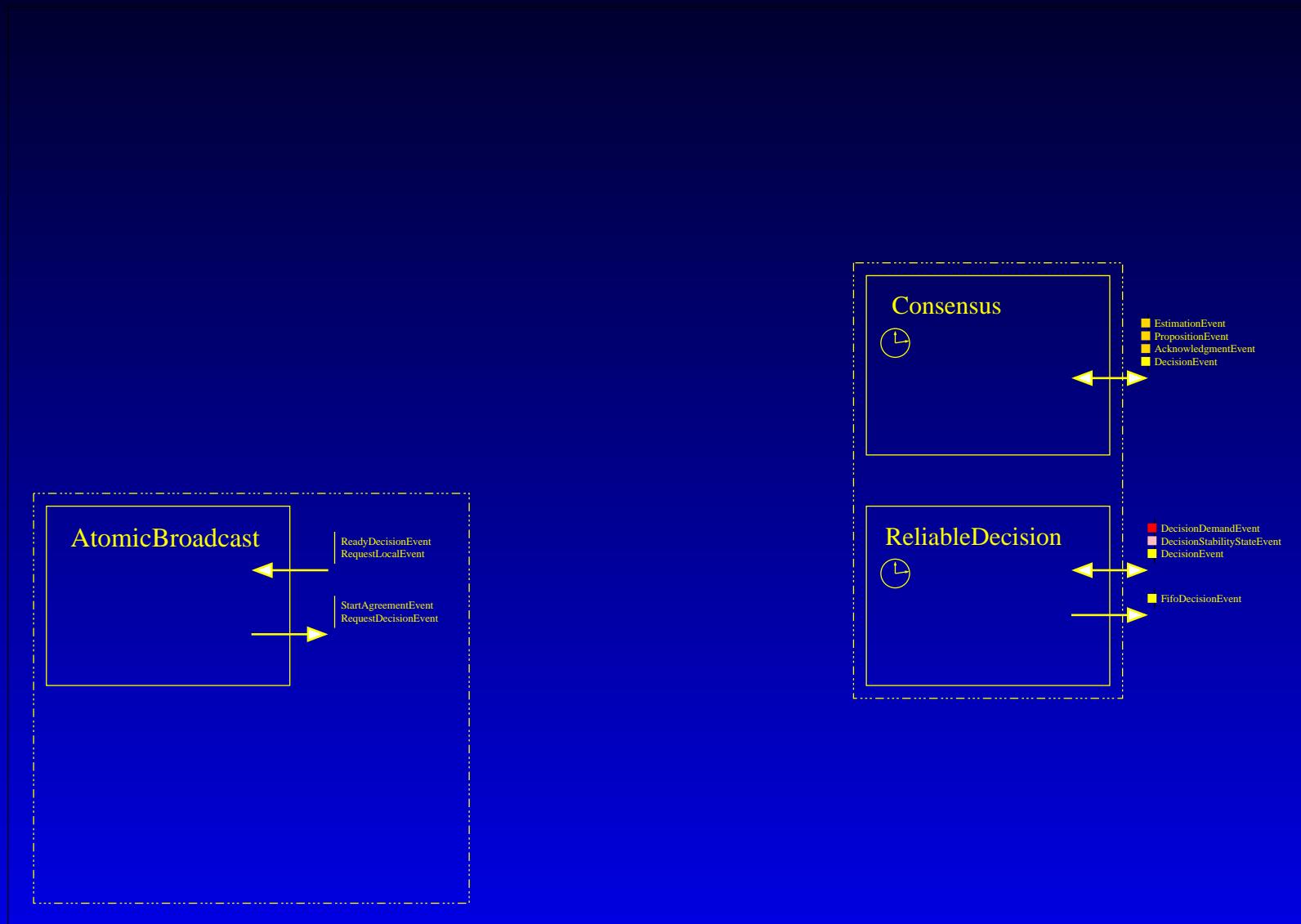
# Overview of Adam



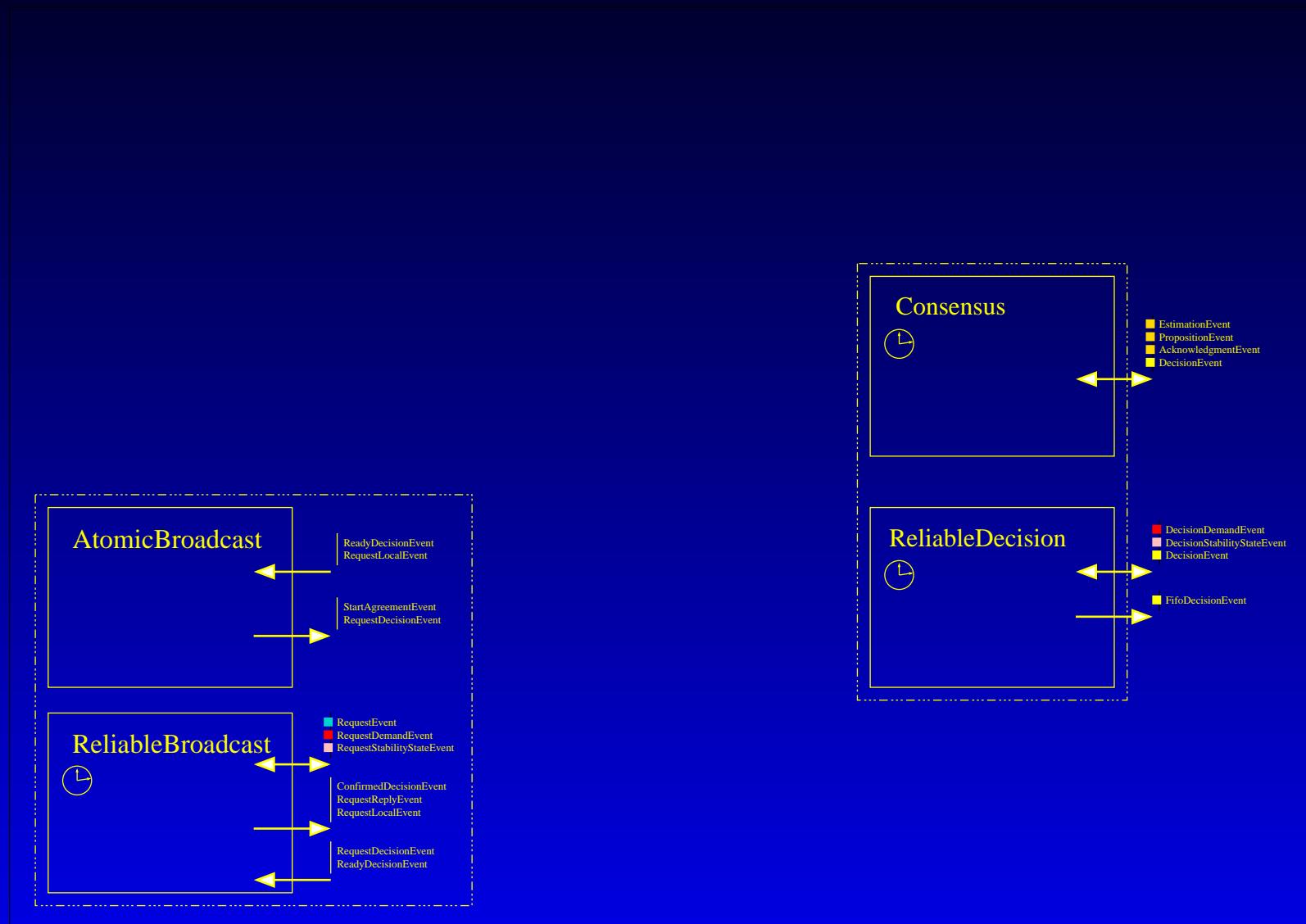
# Overview of Adam



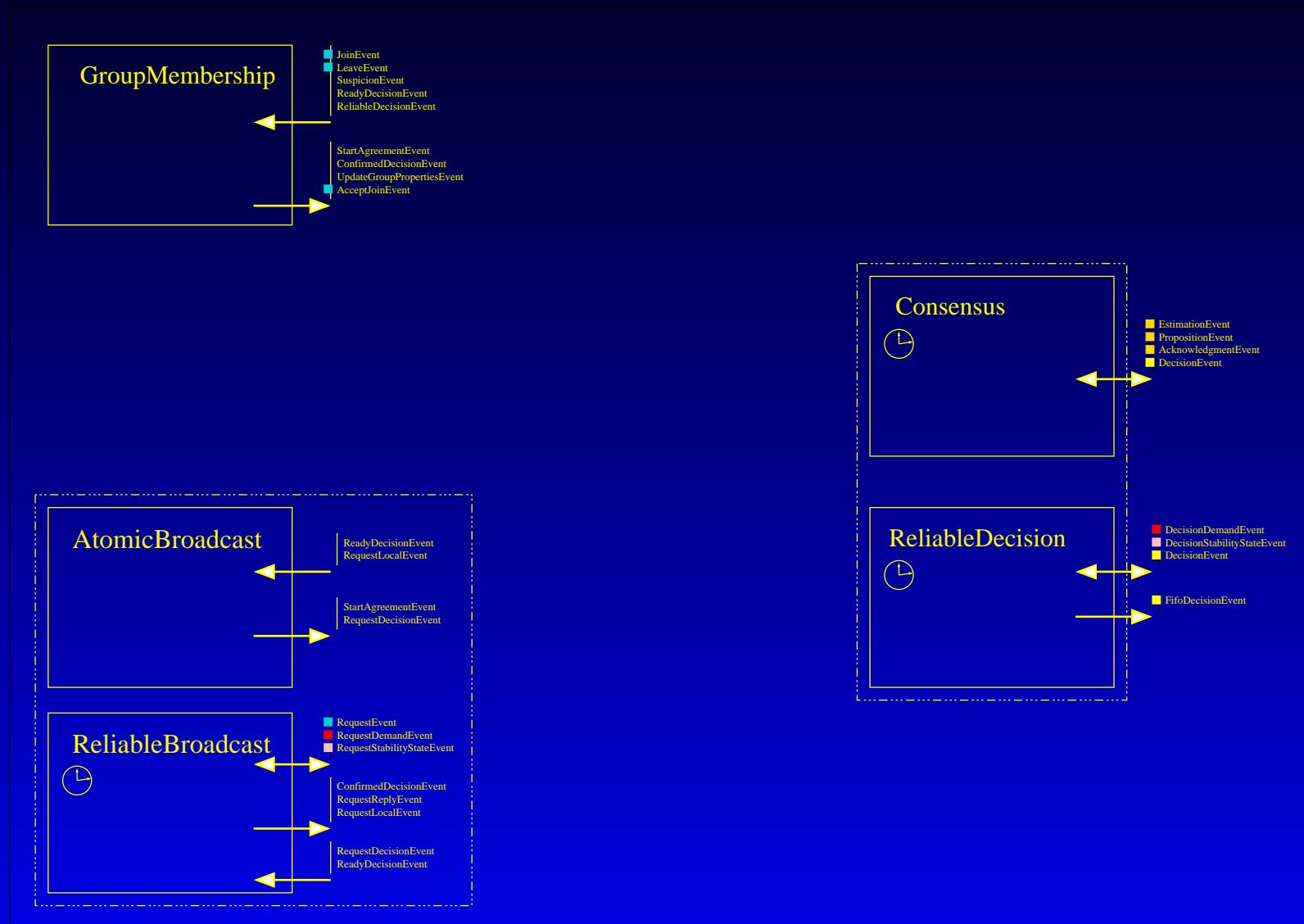
# Overview of Adam



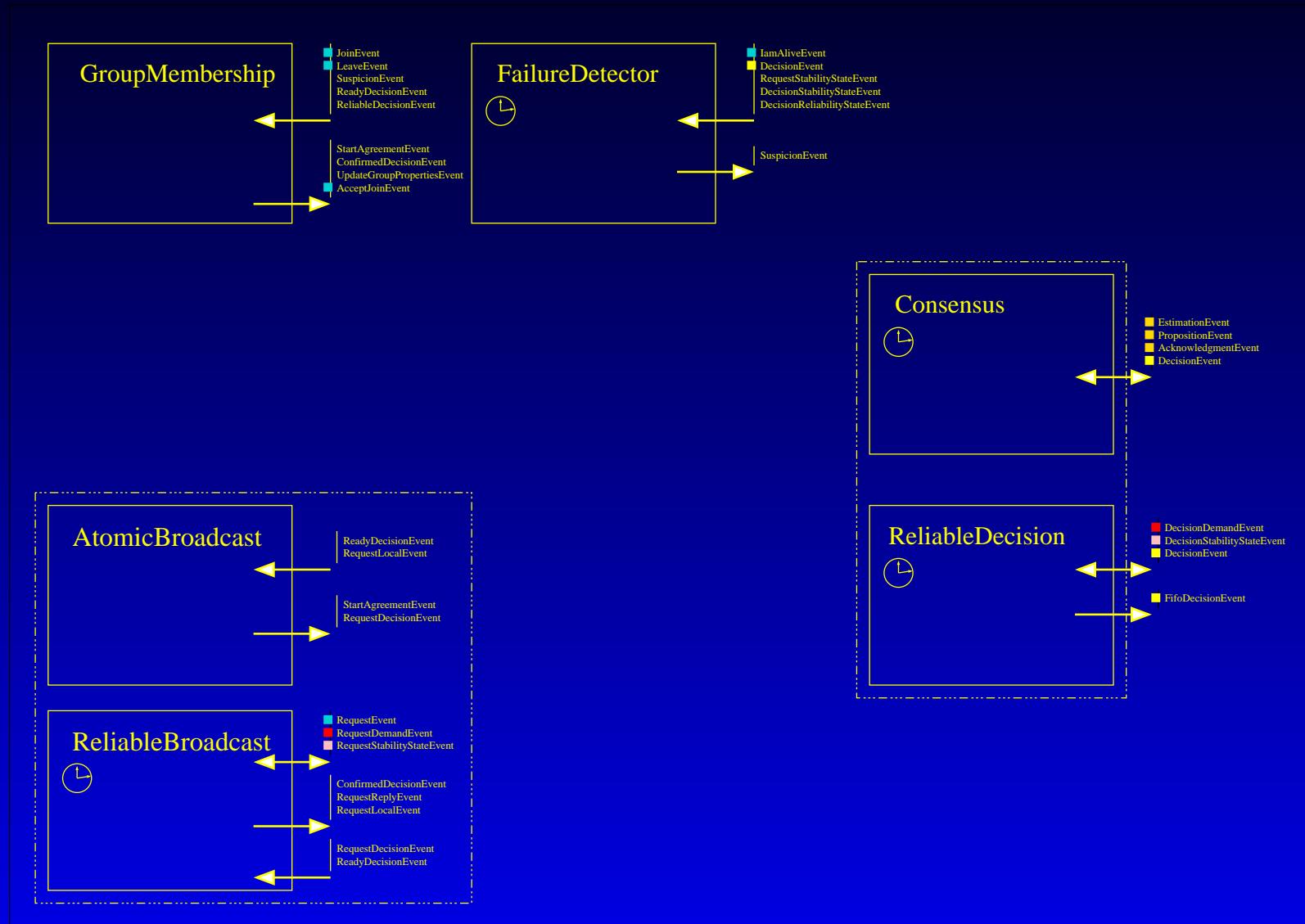
# Overview of Adam



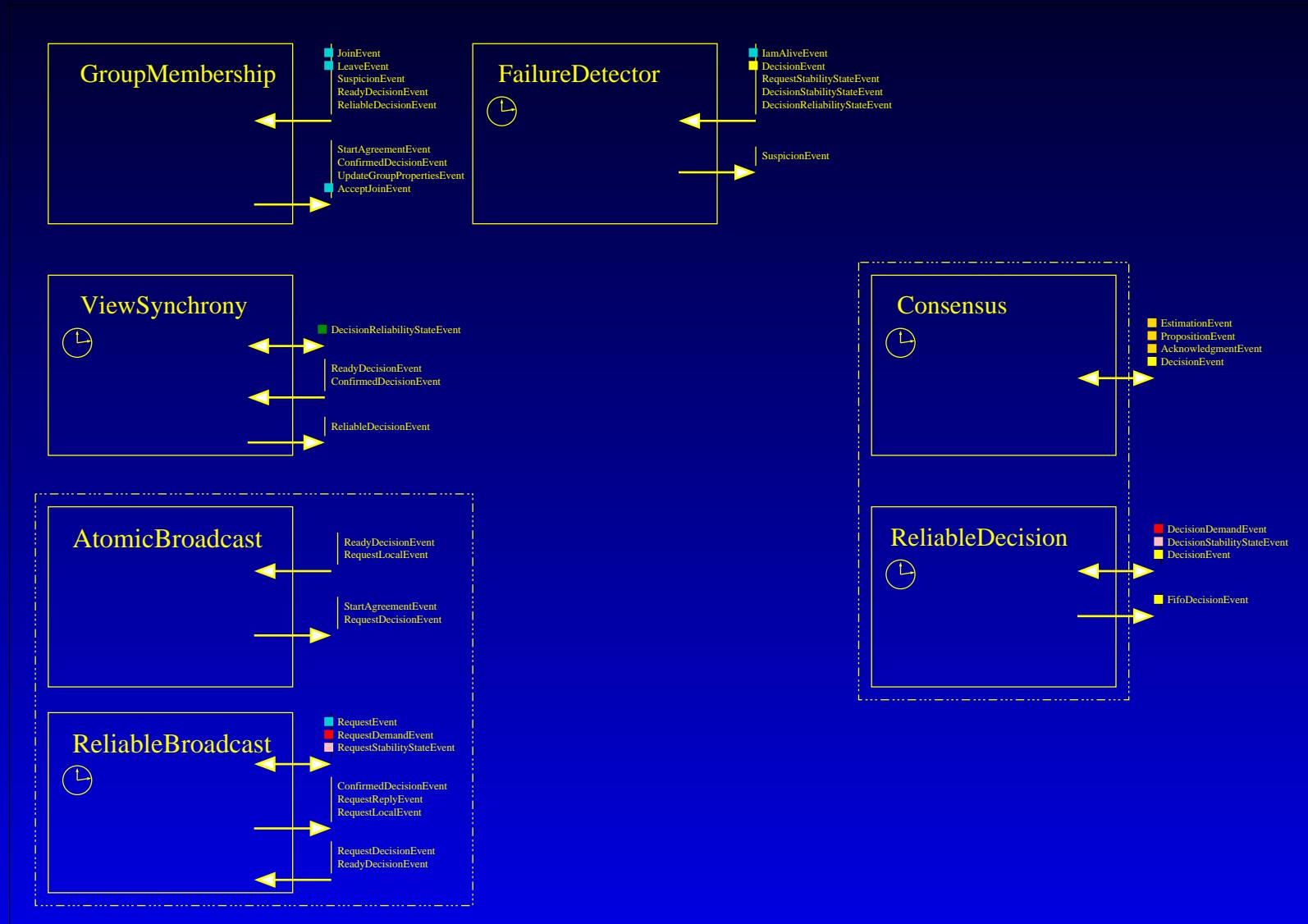
# Overview of Adam



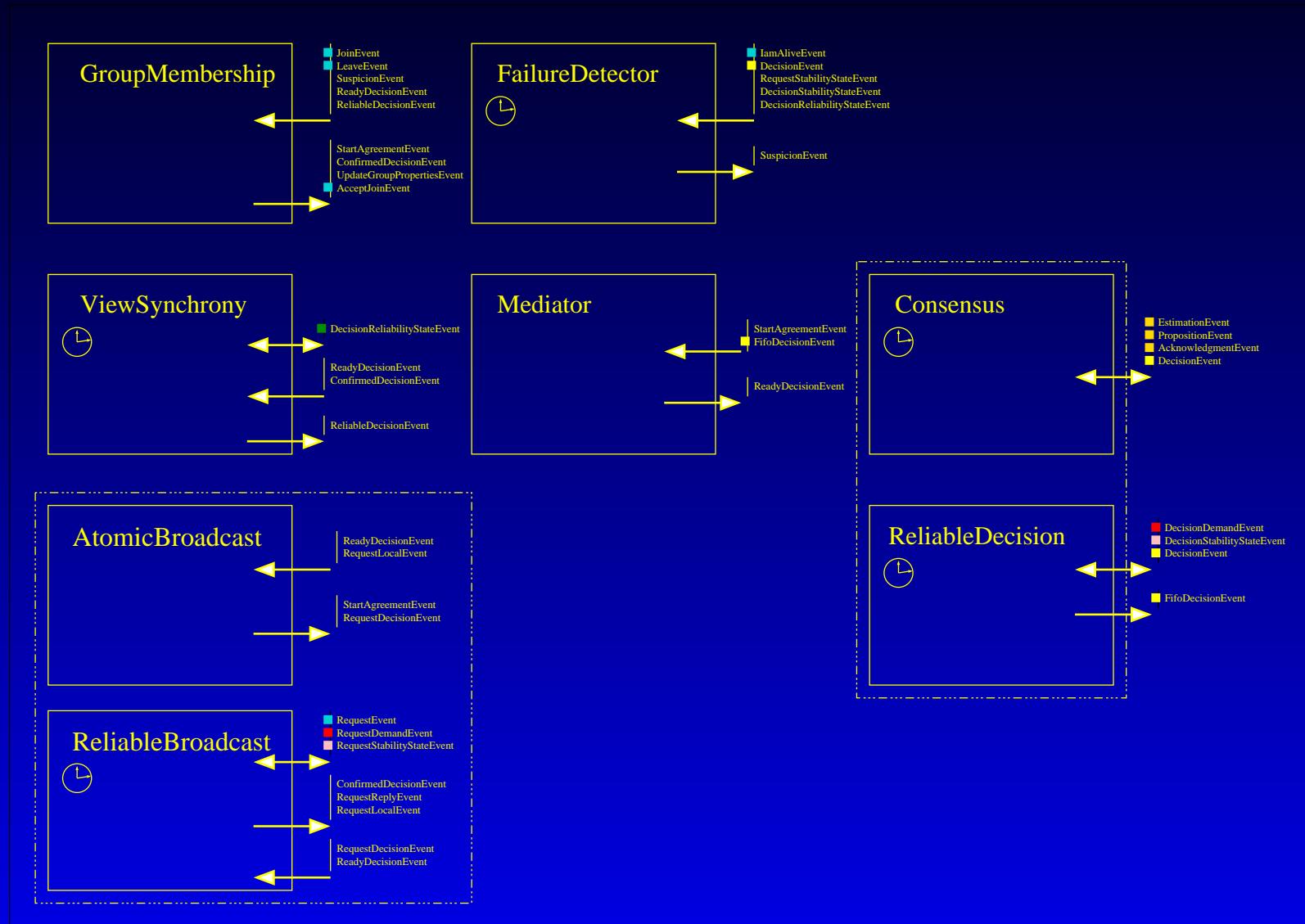
# Overview of Adam



# Overview of Adam



# Overview of Adam



# The GAF Framework

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.
- It is basically derived from the original Chandra-Toueg algorithm for solving consensus using  $\diamond S$  failure detectors.

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.
- It is basically derived from the original Chandra-Toueg algorithm for solving consensus using  $\diamond S$  failure detectors.
- However, it has been improved to take into account a large spectrum of practical requirements :

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.
- It is basically derived from the original Chandra-Toueg algorithm for solving consensus using  $\diamond S$  failure detectors.
- However, it has been improved to take into account a large spectrum of practical requirements :
  - Versatility

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.
- It is basically derived from the original Chandra-Toueg algorithm for solving consensus using  $\diamond S$  failure detectors.
- However, it has been improved to take into account a large spectrum of practical requirements :
  - Versatility
  - Efficiency

# The GAF Framework

- GAF framework is the synthesis of about four years of research in the field of agreement problems applied to fault-tolerance.
- It is basically derived from the original Chandra-Toueg algorithm for solving consensus using  $\diamond S$  failure detectors.
- However, it has been improved to take into account a large spectrum of practical requirements :
  - Versatility
  - Efficiency
  - Dynamical Adaptation to the Network Conditions

# The GAF Framework

# The GAF Framework

- GAF uses the rotating coordinator paradigm like in CT algorithm.

# The GAF Framework

- GAF uses the rotating coordinator paradigm like in CT algorithm.
- The algorithm progresses by round.

# The GAF Framework

- GAF uses the rotating coordinator paradigm like in CT algorithm.
- The algorithm progresses by round.
- Each round is coordinated by a unique coordinator determined by the round number.

# The GAF Framework

- GAF uses the rotating coordinator paradigm like in CT algorithm.
- The algorithm progresses by round.
- Each round is coordinated by a unique coordinator determined by the round number.
- The coordinator tries to collect a majority of estimations values of the final decision during the round.

# The GAF Framework

- GAF uses the rotating coordinator paradigm like in CT algorithm.
- The algorithm progresses by round.
- Each round is coordinated by a unique coordinator determined by the round number.
- The coordinator tries to collect a majority of estimations values of the final decision during the round.
- In a second phase, if it has been able to collect such a majority, it computes and broadcast a proposition.

# The GAF Framework

- The algorithm progresses by round.
- Each round is coordinated by a unique coordinator determined by the round number.
- The coordinator tries to collect a majority of estimations values of the final decision during the round.
- In a second phase, if it has been able to collect such a majority, it computes and broadcast a proposition.
- To become the final decision, this proposition must be received and acknowledged by a majority of processes.

# The GAF Framework

- Each round is coordinated by a unique coordinator determined by the round number.
- The coordinator tries to collect a majority of estimations values of the final decision during the round.
- In a second phase, if it has been able to collect such a majority, it computes and broadcast a proposition.
- To become the final decision, this proposition must be received and acknowledged by a majority of processes.
- A process that receives the proposition from the coordinator, retains it for its new estimation.

# The GAF Framework

- The coordinator tries to collect a majority of estimations values of the final decision during the round.
- In a second phase, if it has been able to collect such a majority, it computes and broadcast a proposition.
- To become the final decision, this proposition must be received and acknowledged by a majority of processes.
- A process that receives the proposition from the coordinator, retains it for its new estimation.
- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.

# The GAF Framework

- In a second phase, if it has been able to collect such a majority, it computes and broadcast a proposition.
- To become the final decision, this proposition must be received and acknowledged by a majority of processes.
- A process that receives the proposition from the coordinator, retains it for its new estimation.
- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.
- Initially, estimations are tagged with 0.

# The GAF Framework

- To become the final decision, this proposition must be received and acknowledged by a majority of processes.
- A process that receives the proposition from the coordinator, retains it for its new estimation.
- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.
- Initially, estimations are tagged with 0.
- We have improved the original CT algorithm by :

# The GAF Framework

- A process that receives the proposition from the coordinator, retains it for its new estimation.
- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.
- Initially, estimations are tagged with 0.
- We have improved the original CT algorithm by :
  - We do not use  $\diamond S$  failure detectors, instead we are using fixed duration rounds.

# The GAF Framework

- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.
- Initially, estimations are tagged with 0.
- We have improved the original CT algorithm by :
  - We do not use  $\diamond \mathcal{S}$  failure detectors, instead we are using fixed duration rounds.
  - This removes the necessity of reliable channels,

# The GAF Framework

- To avoid concurrent and different decisions, each estimation is tagged with the round during which it was retained.
- Initially, estimations are tagged with 0.
- We have improved the original CT algorithm by :
  - We do not use  $\diamond \mathcal{S}$  failure detectors, instead we are using fixed duration rounds.
  - This removes the necessity of reliable channels, while at the same time requiring a good synchronization between clocks.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We do not use  $\diamond \mathcal{S}$  failure detectors, instead we are using fixed duration rounds.
  - This removes the necessity of reliable channels, while at the same time requiring a good synchronization between clocks.
  - To relax this assumption, we are using a window mechanism.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We do not use  $\diamond\mathcal{S}$  failure detectors, instead we are using fixed duration rounds.
  - This removes the necessity of reliable channels, while at the same time requiring a good synchronization between clocks.
  - To relax this assumption, we are using a window mechanism.
  - This enables the coordinator of a round to take into account messages coming from both the future and the past.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We take advantage of the fact that the coordinator collects a majority of estimations.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We take advantage of the fact that the coordinator collects a majority of estimations.
  - It can compute a more significant proposition by mixing all the estimations when they are all tagged with 0.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We take advantage of the fact that the coordinator collects a majority of estimations.
  - It can compute a more significant proposition by mixing all the estimations when they are all tagged with 0.
  - Processes are allowed to propose empty values.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We take advantage of the fact that the coordinator collects a majority of estimations.
  - It can compute a more significant proposition by mixing all the estimations when they are all tagged with 0.
  - Processes are allowed to propose empty values.
  - As long as, no non empty proposition from the coordinator has been made, they can improve their estimations.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We take advantage of the fact that the coordinator collects a majority of estimations.
  - It can compute a more significant proposition by mixing all the estimations when they are all tagged with 0.
  - Processes are allowed to propose empty values.
  - As long as, no non empty proposition from the coordinator has been made, they can improve their estimations.
  - This has proven to be very efficient to improve the throughputness of atomic broadcast.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We allows the programmer to solve multiple agreement problems at the same time.

# The GAF Framework

- We have improved the original CT algorithm by :
  - We allows the programmer to solve multiple agreement problems at the same time.
  - To do this, each decision taken by GAF is a vector of subdecisions. One entry by agreement problems to solve.

# The GAF Framework

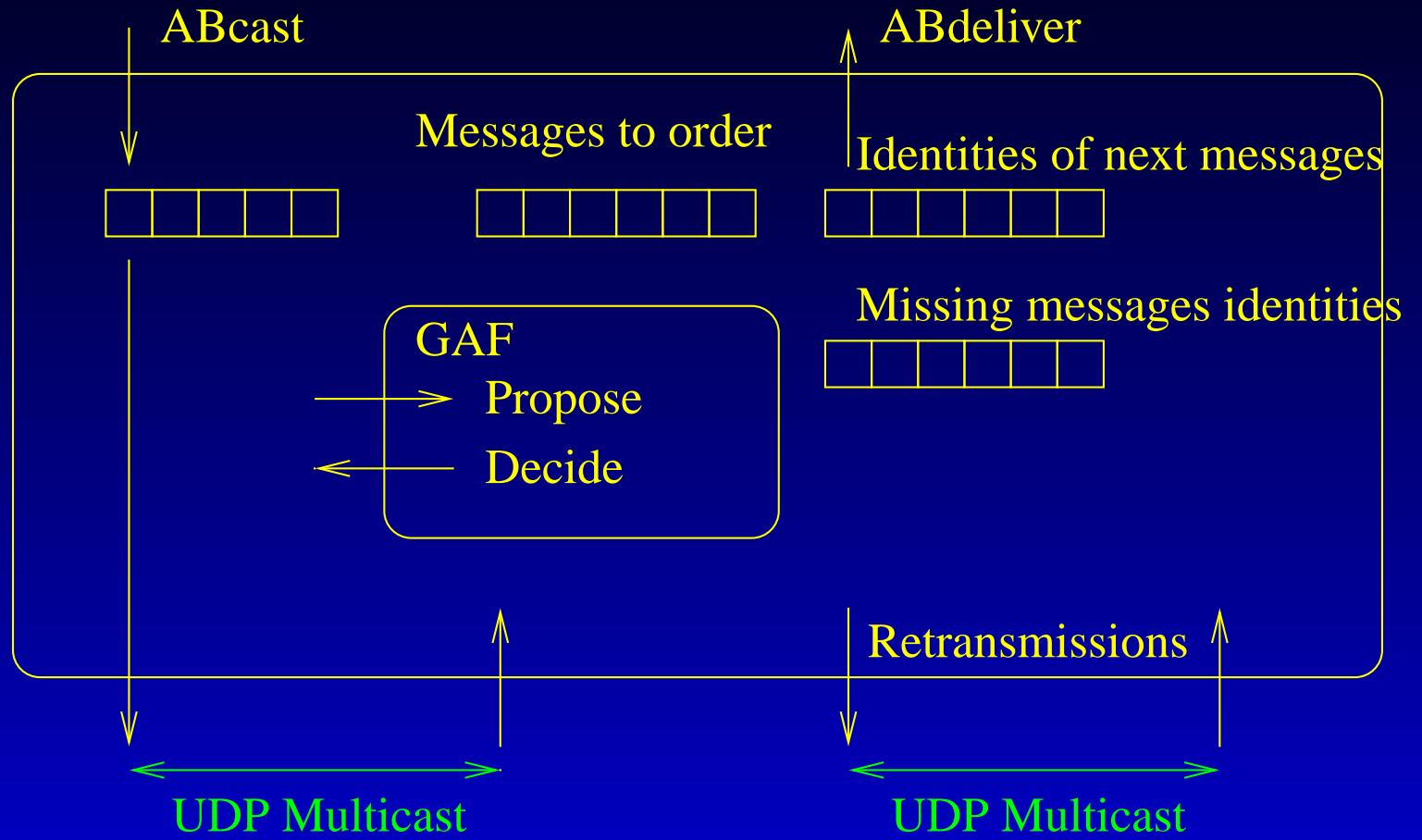
- We have improved the original CT algorithm by :
  - We allows the programmer to solve multiple agreement problems at the same time.
  - To do this, each decision taken by GAF is a vector of subdecisions. One entry by agreement problems to solve.
  - Some of the decisions, may be empty decisions.

# The GAF Framework

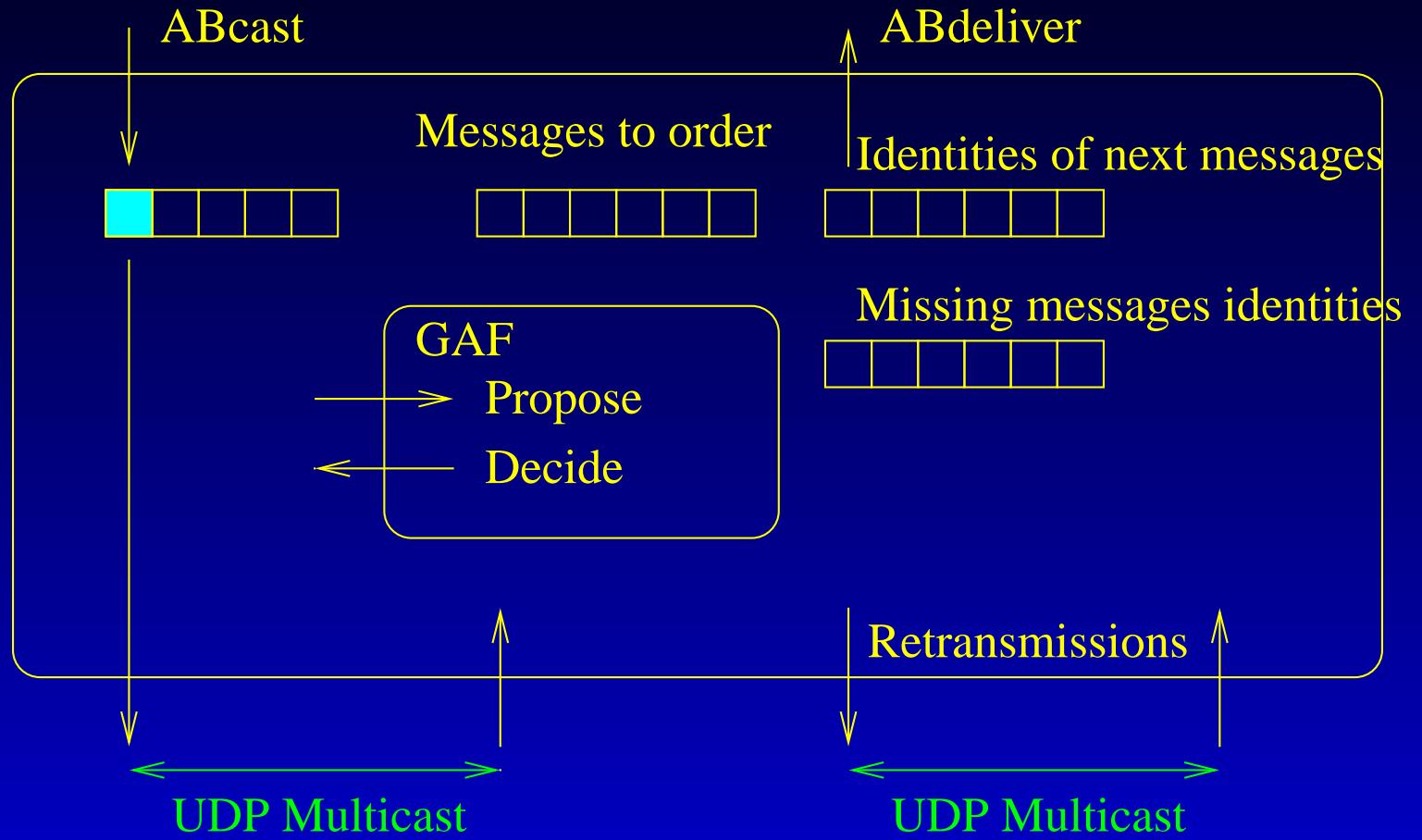
- We have improved the original CT algorithm by :
  - We allows the programmer to solve multiple agreement problems at the same time.
  - To do this, each decision taken by GAF is a vector of subdecisions. One entry by agreement problems to solve.
  - Some of the decisions, may be empty decisions.
  - This is made possible by the previous improvement about multiple initial propositions and empty propositions.

# Solving Atomic Broadcast

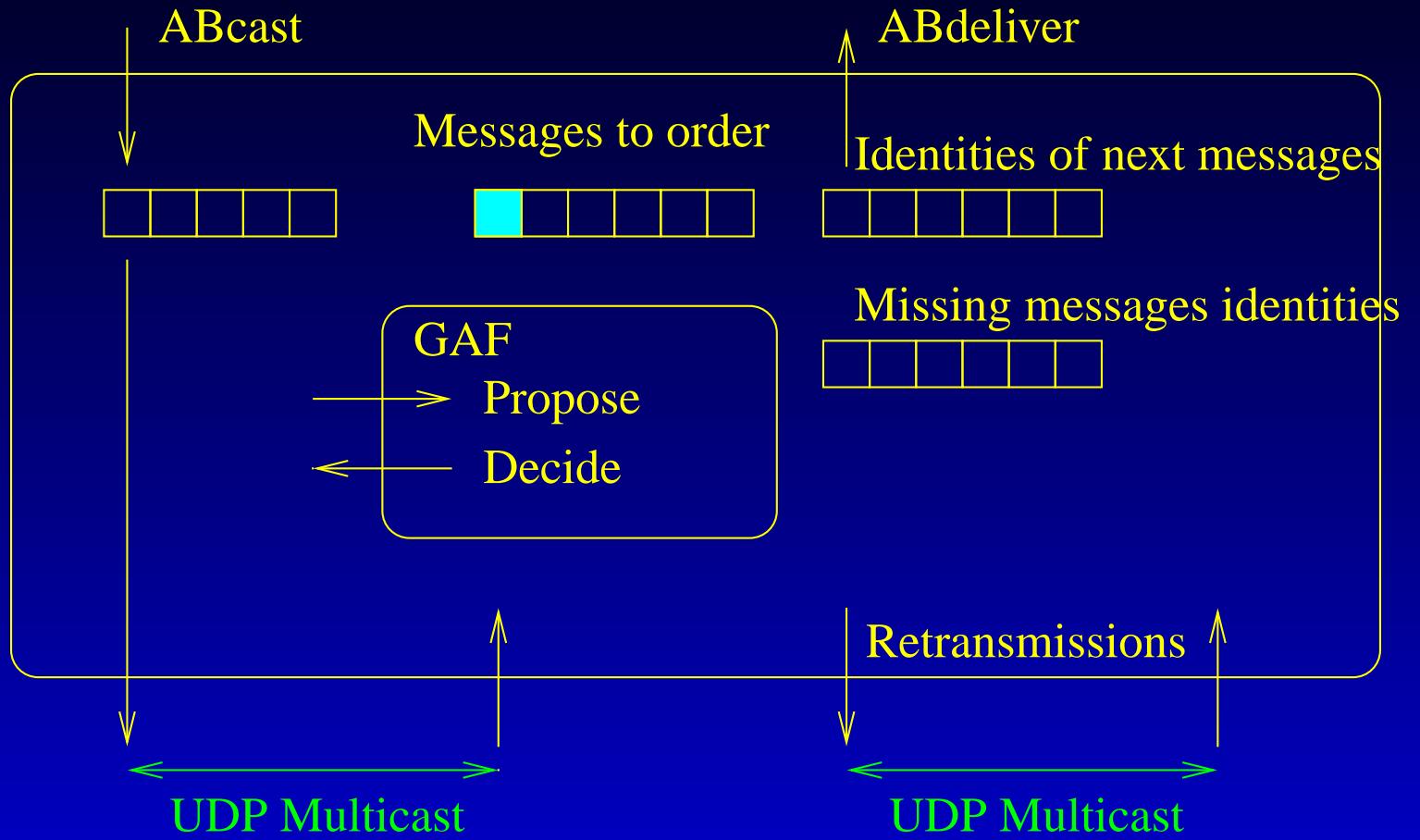
# Solving Atomic Broadcast



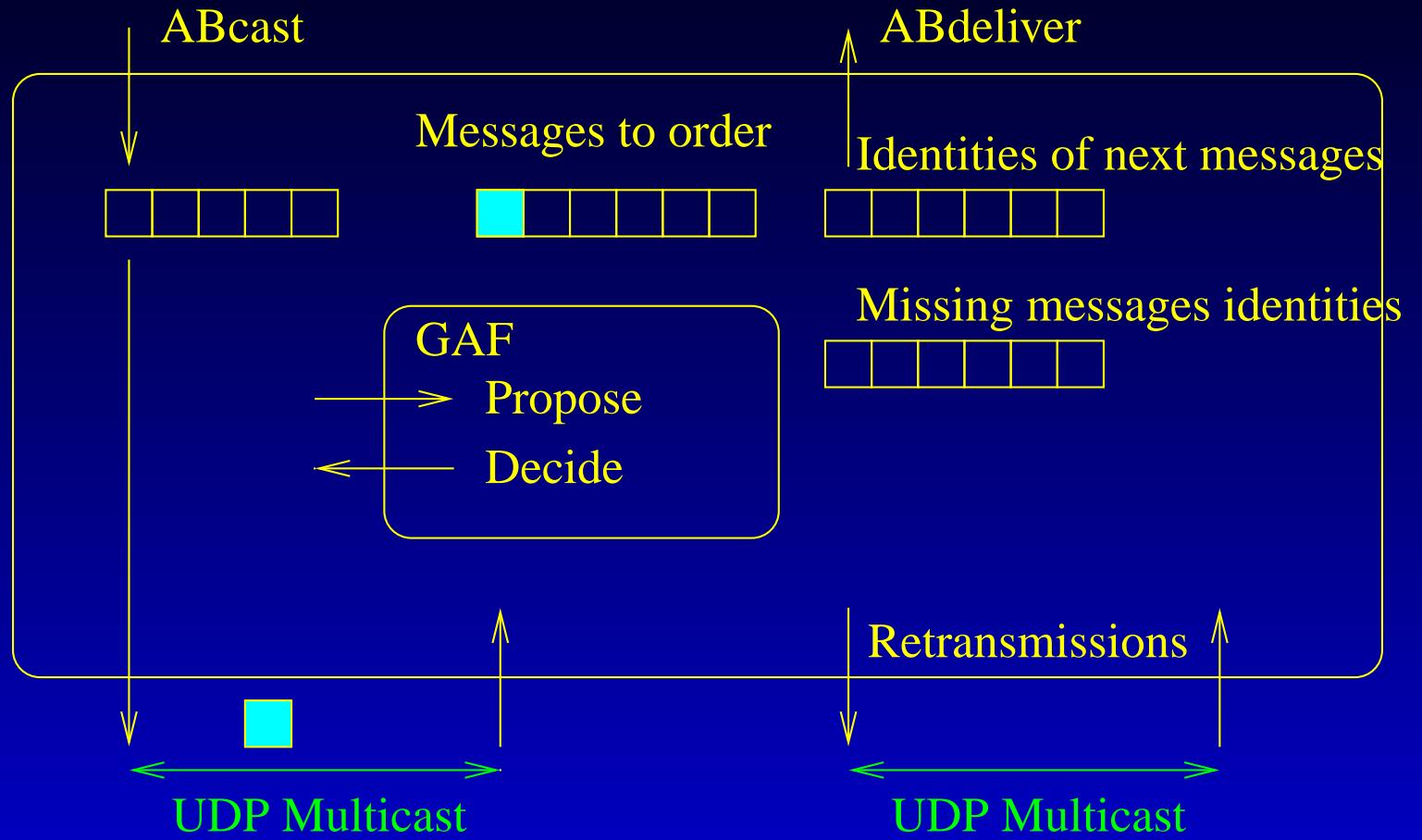
# Solving Atomic Broadcast



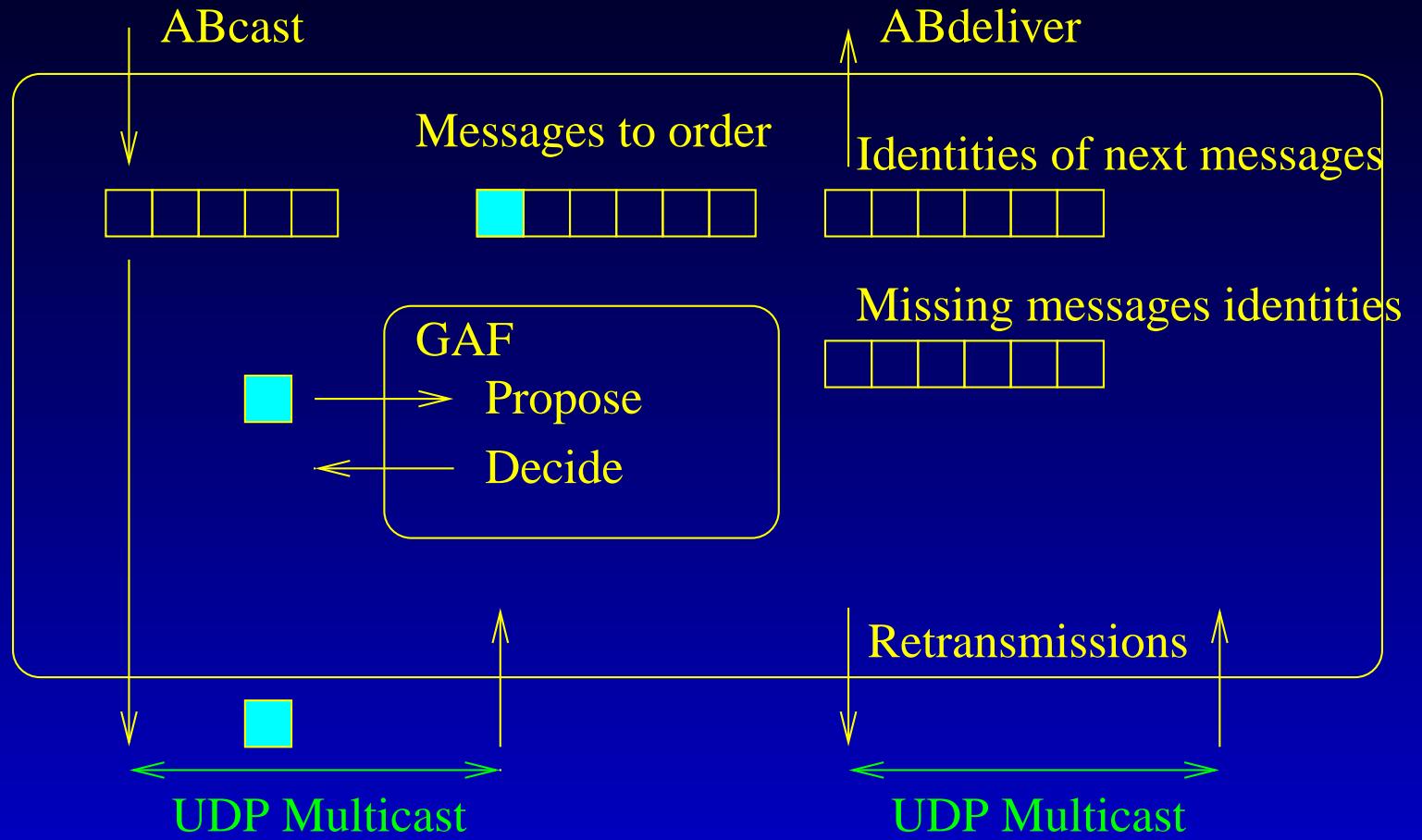
# Solving Atomic Broadcast



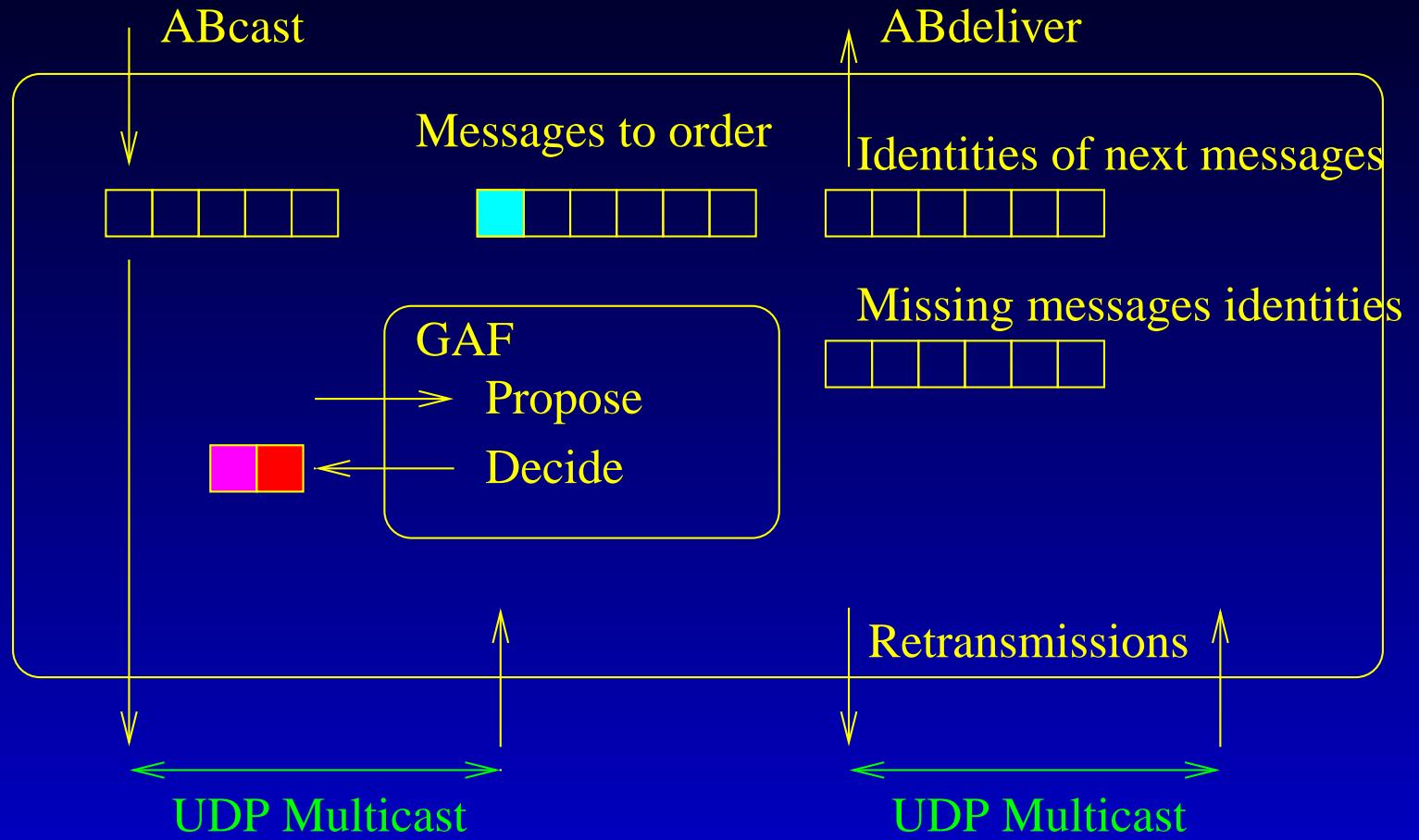
# Solving Atomic Broadcast



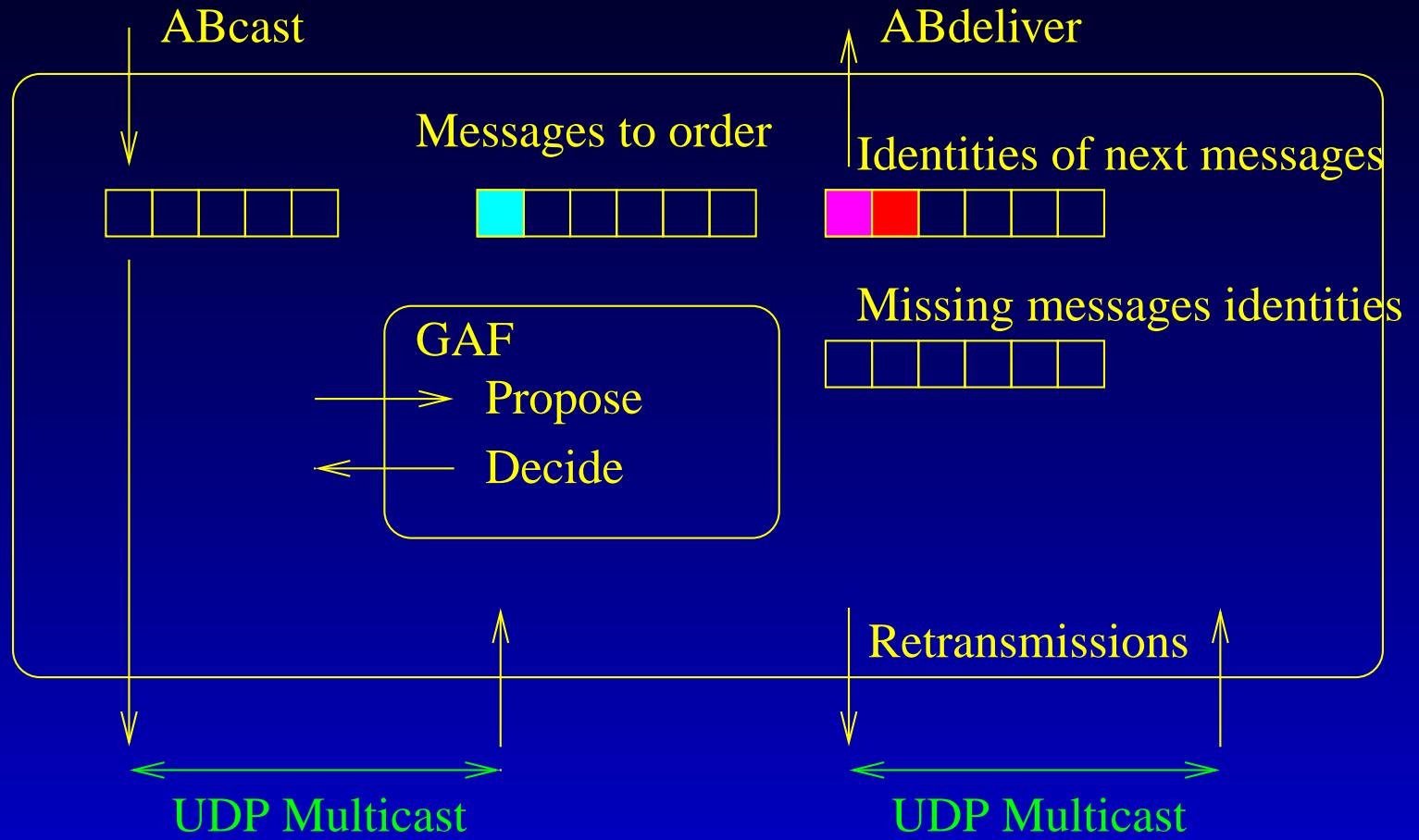
# Solving Atomic Broadcast



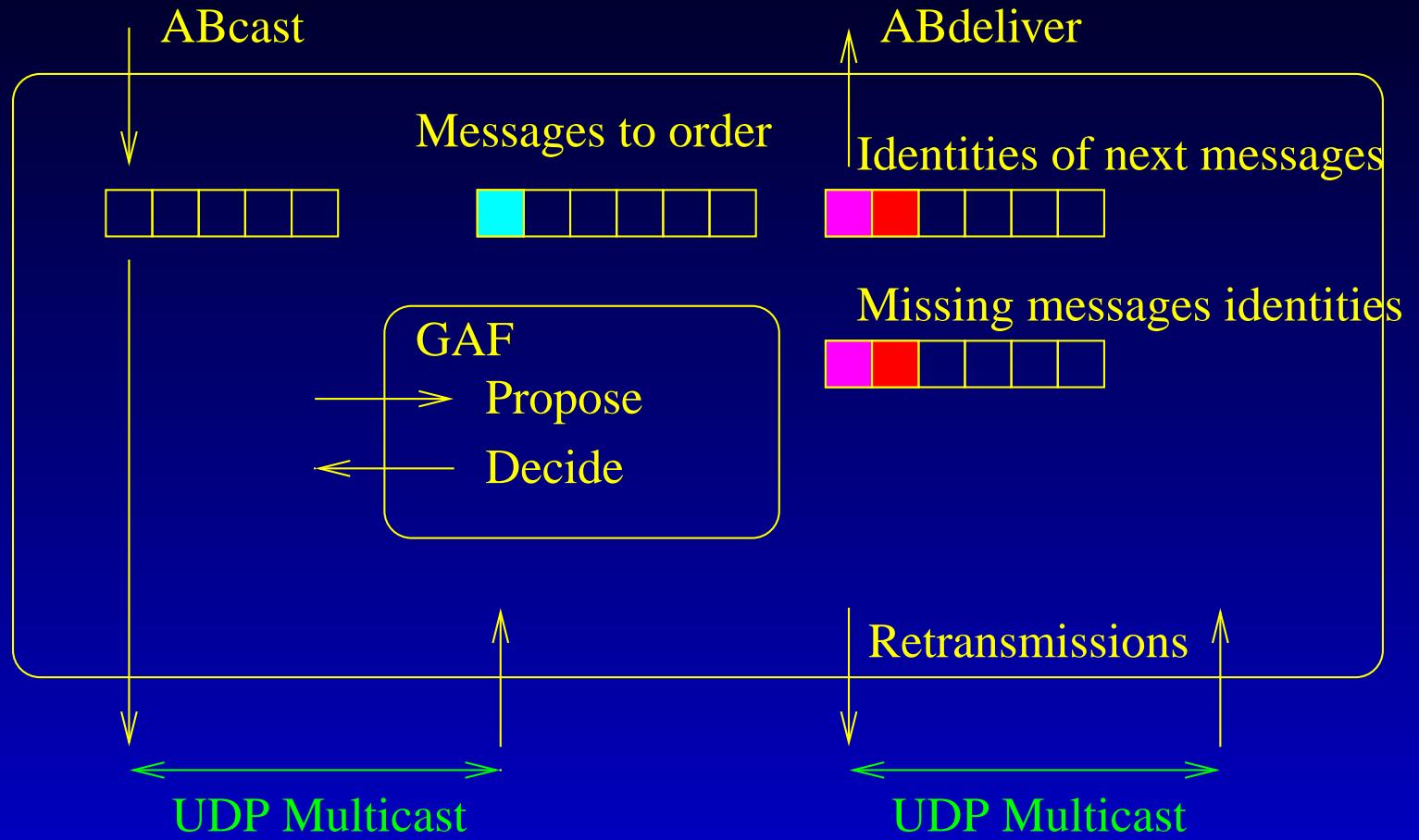
# Solving Atomic Broadcast



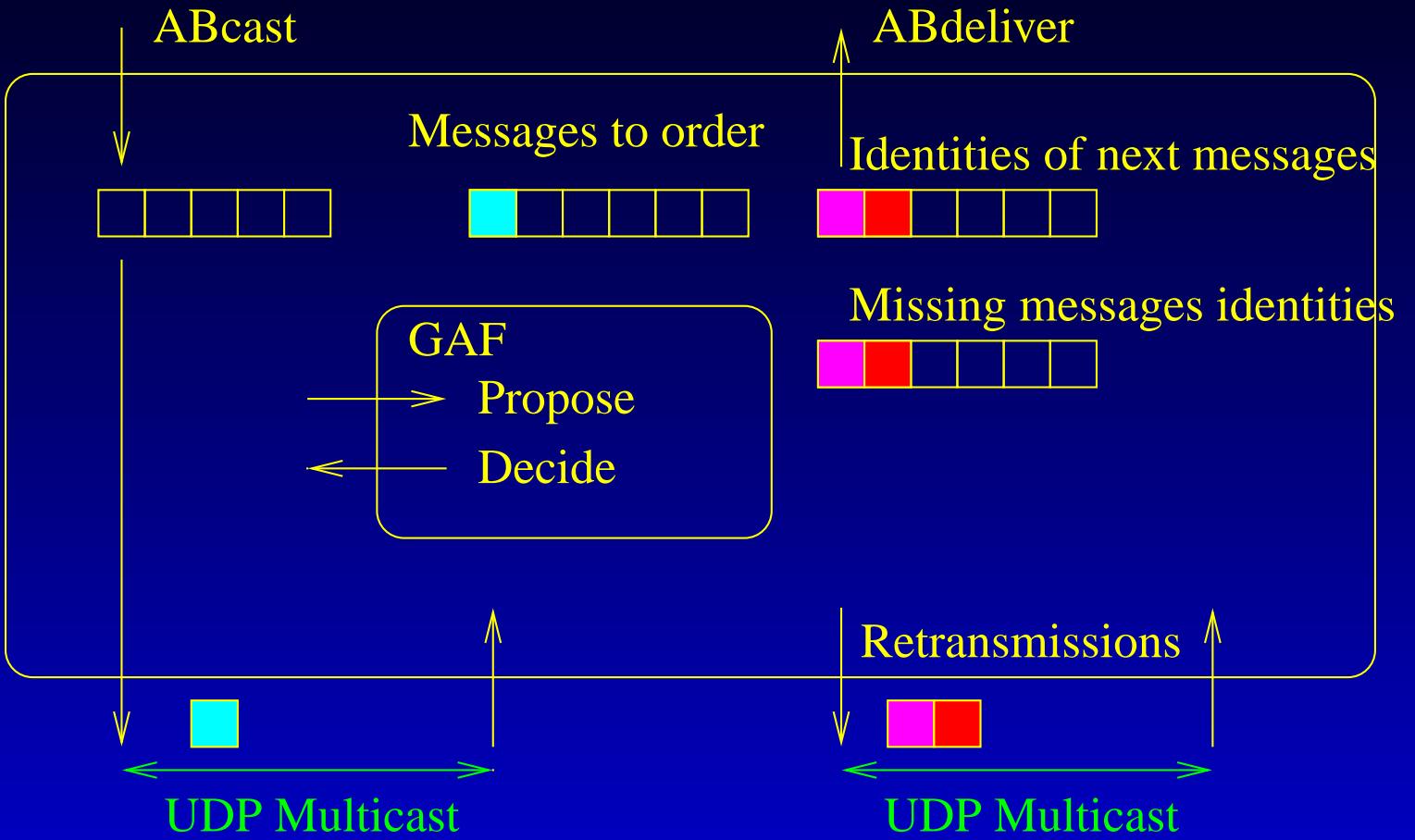
# Solving Atomic Broadcast



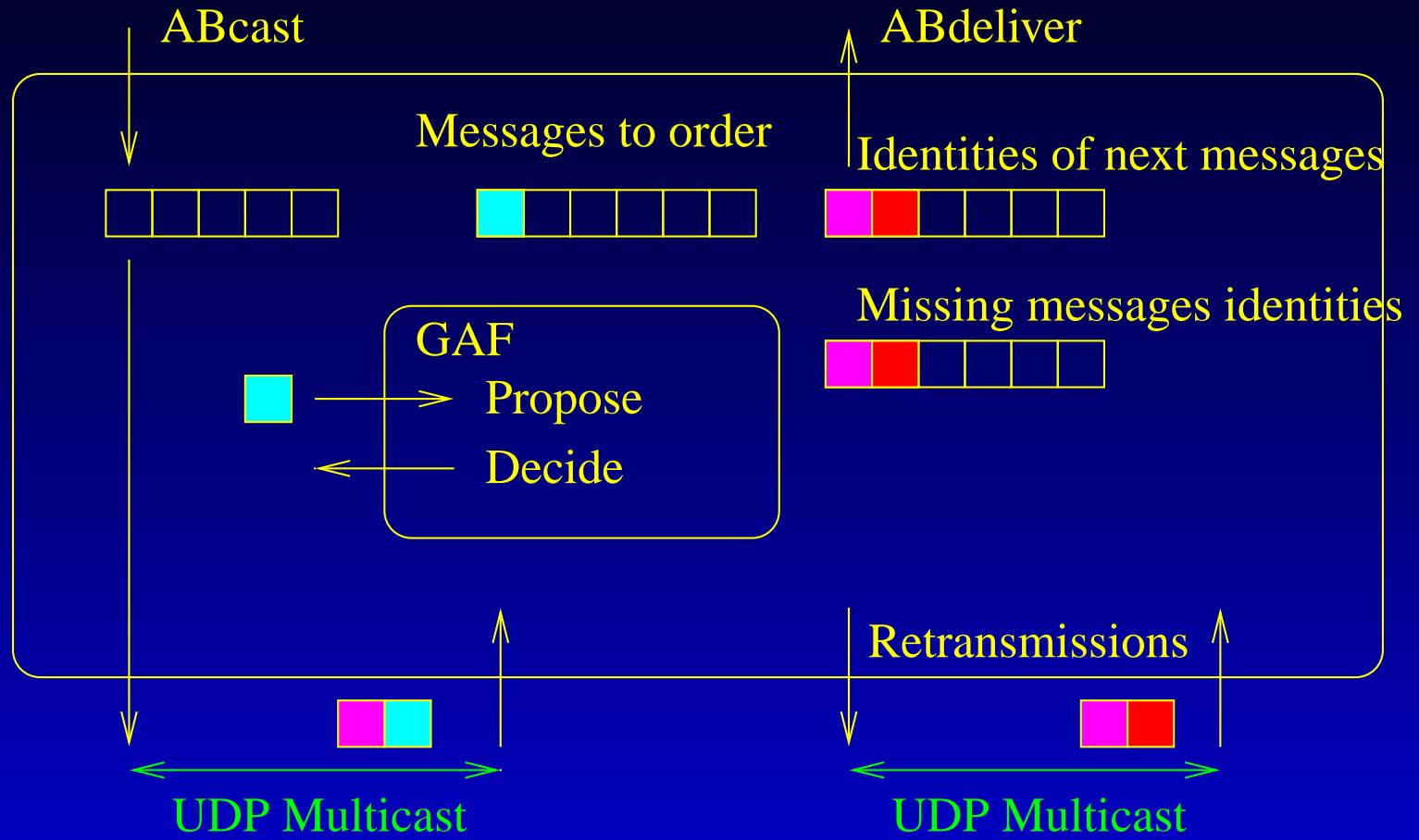
# Solving Atomic Broadcast



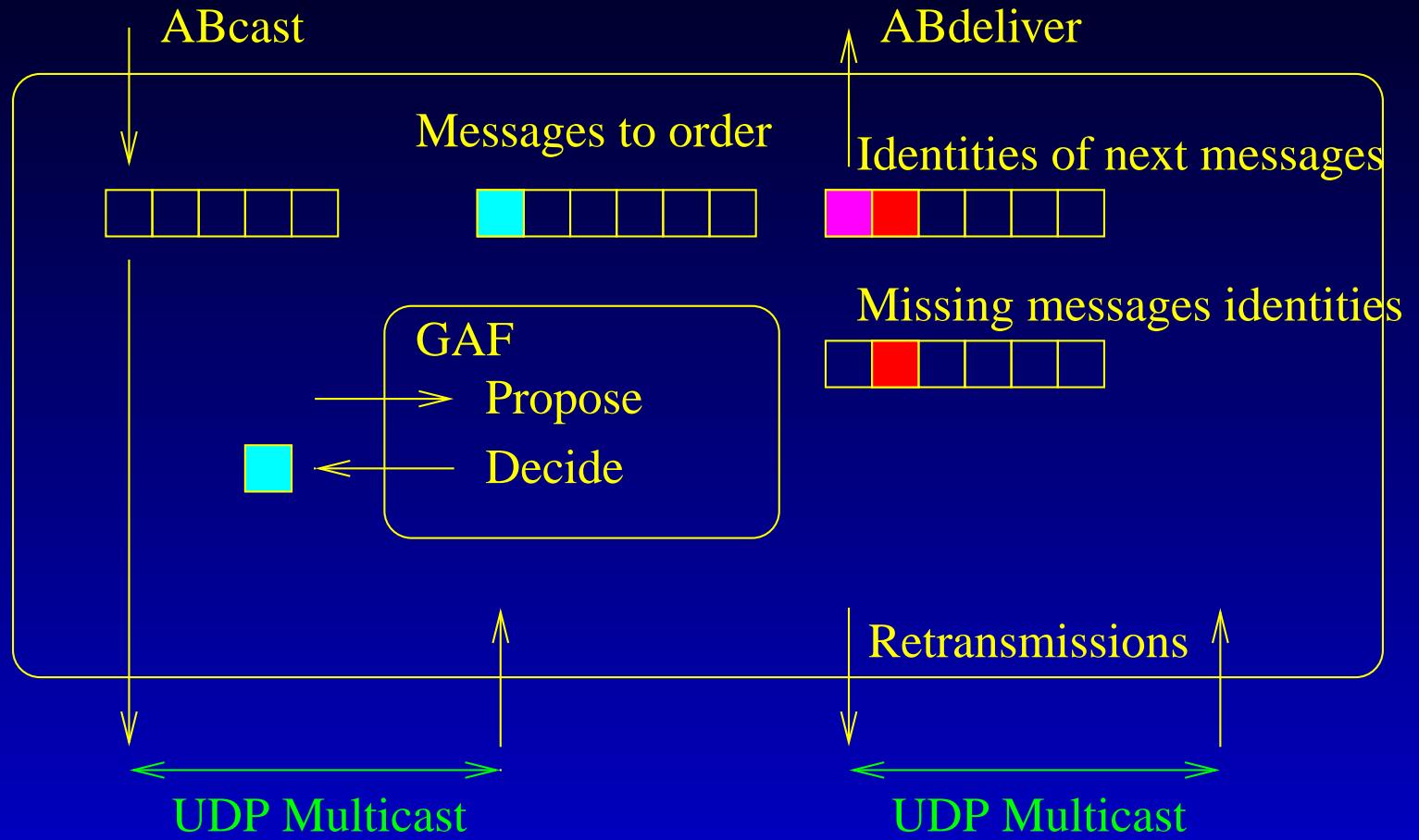
# Solving Atomic Broadcast



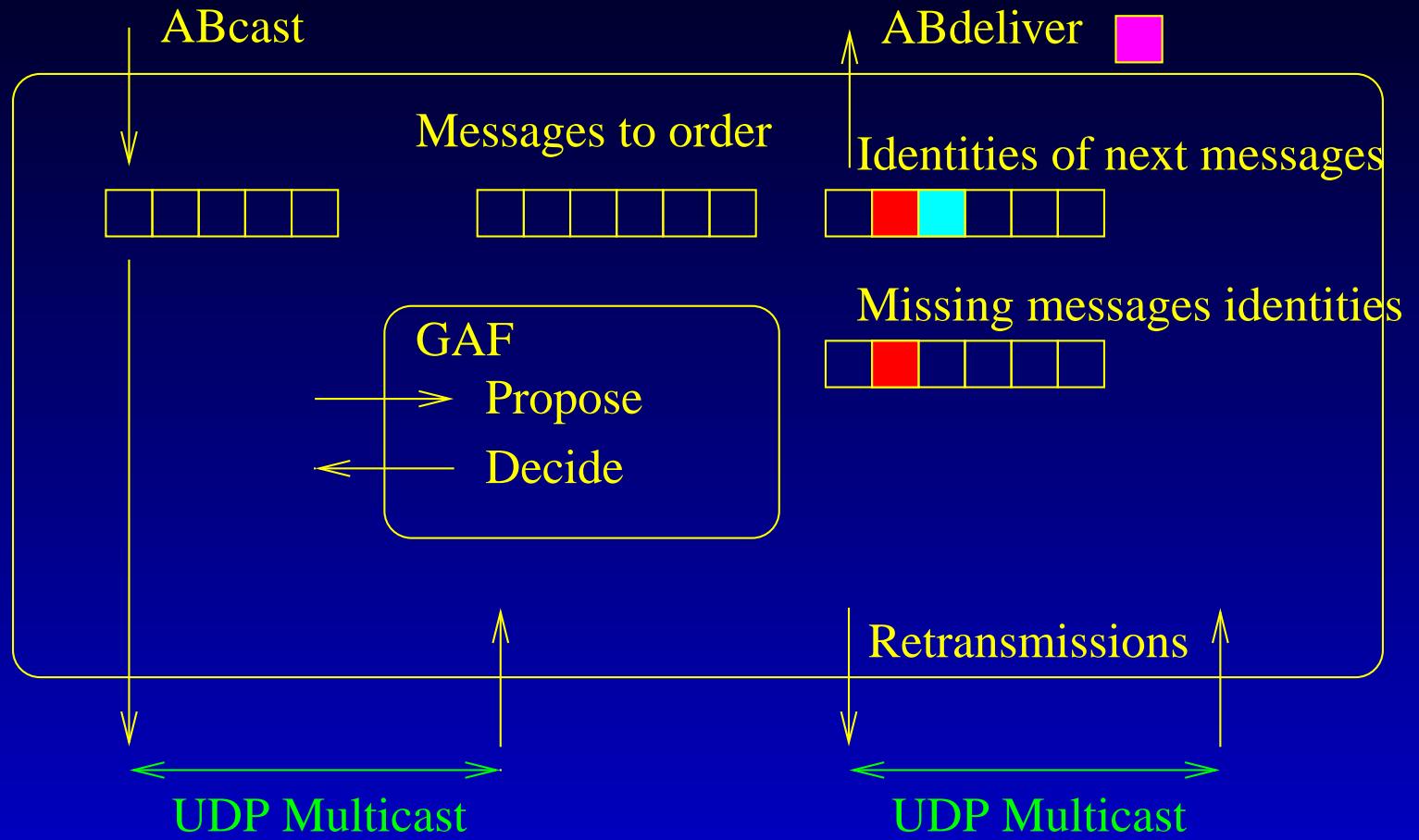
# Solving Atomic Broadcast



# Solving Atomic Broadcast

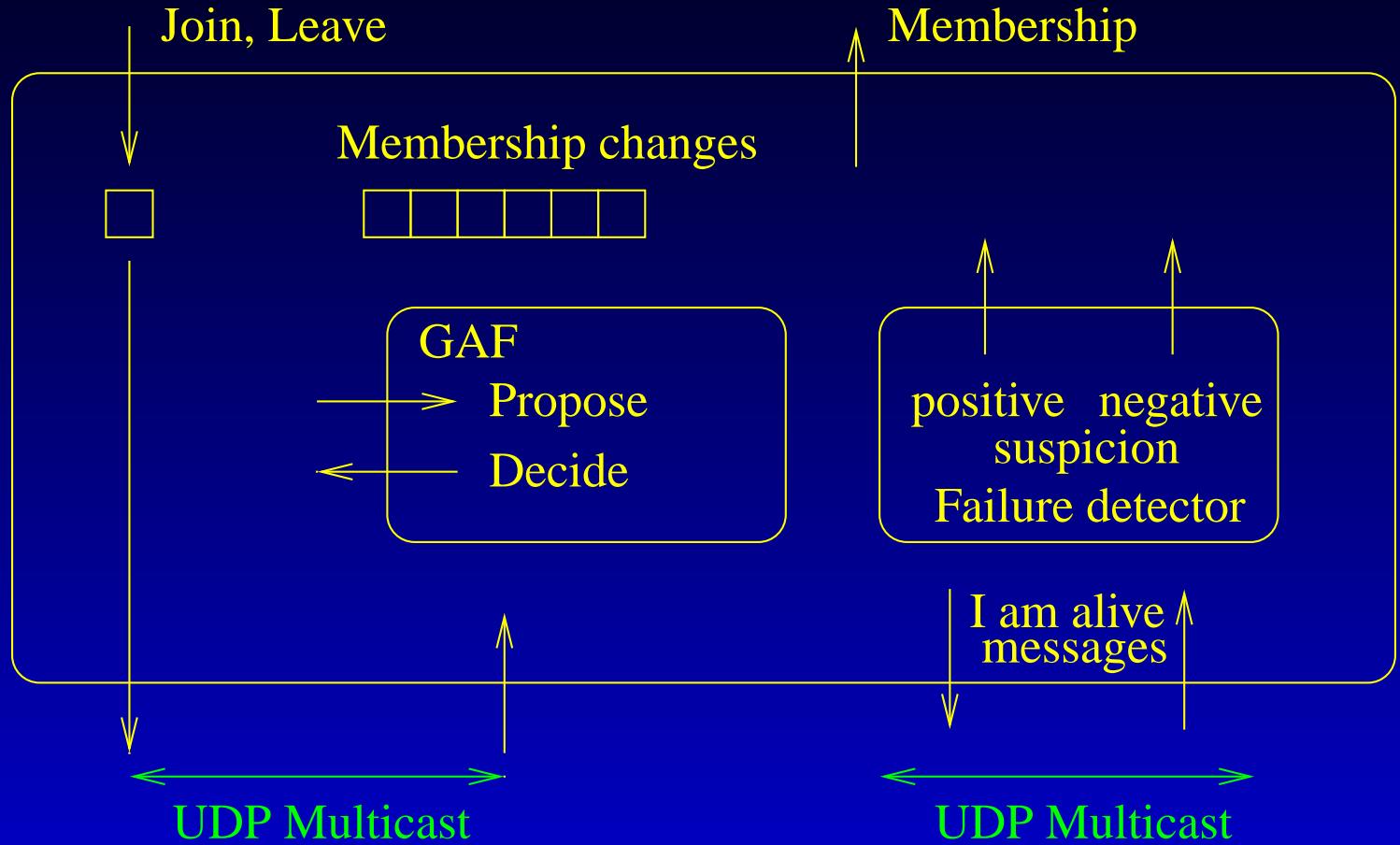


# Solving Atomic Broadcast

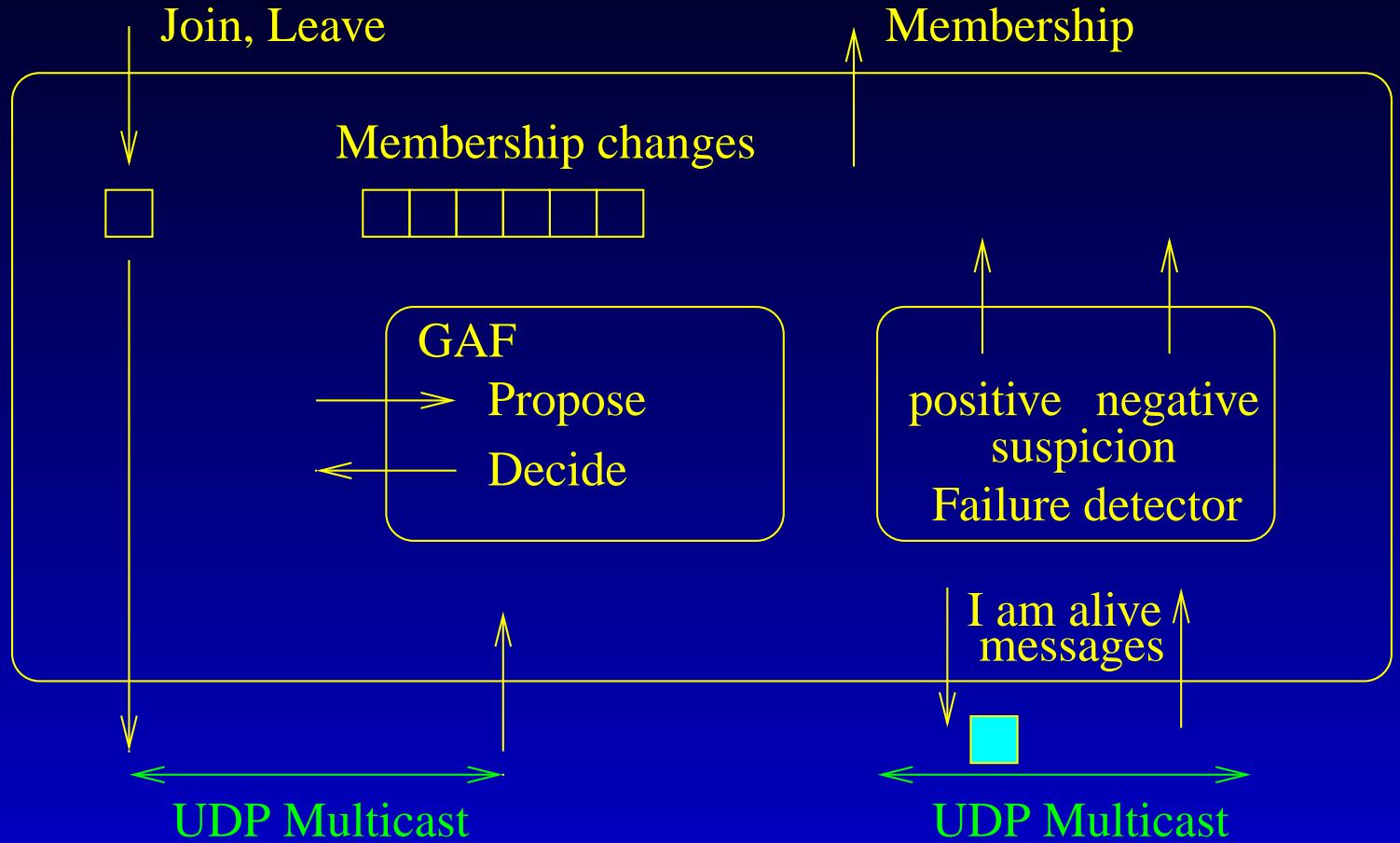


# Solving Membership

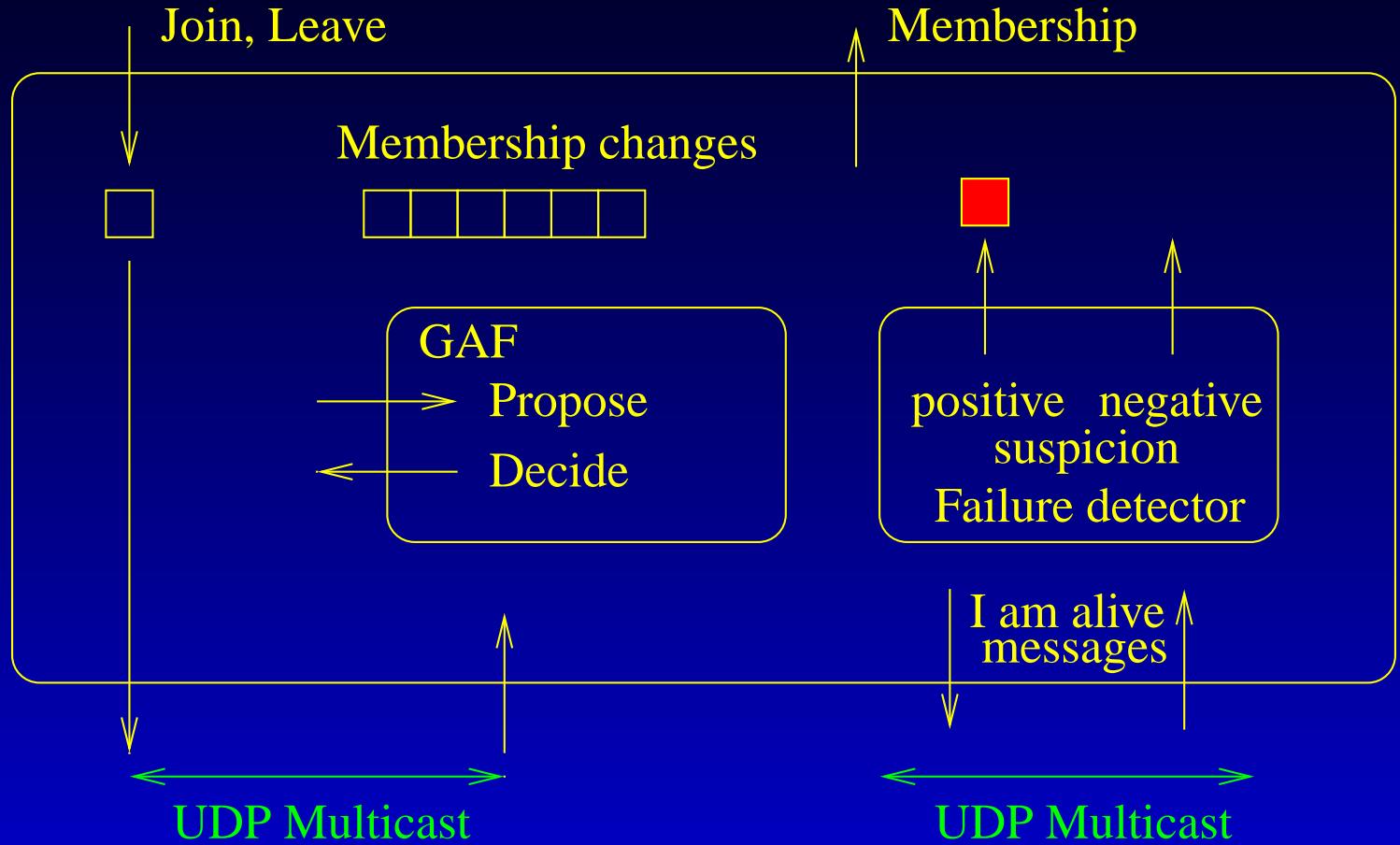
# Solving Membership



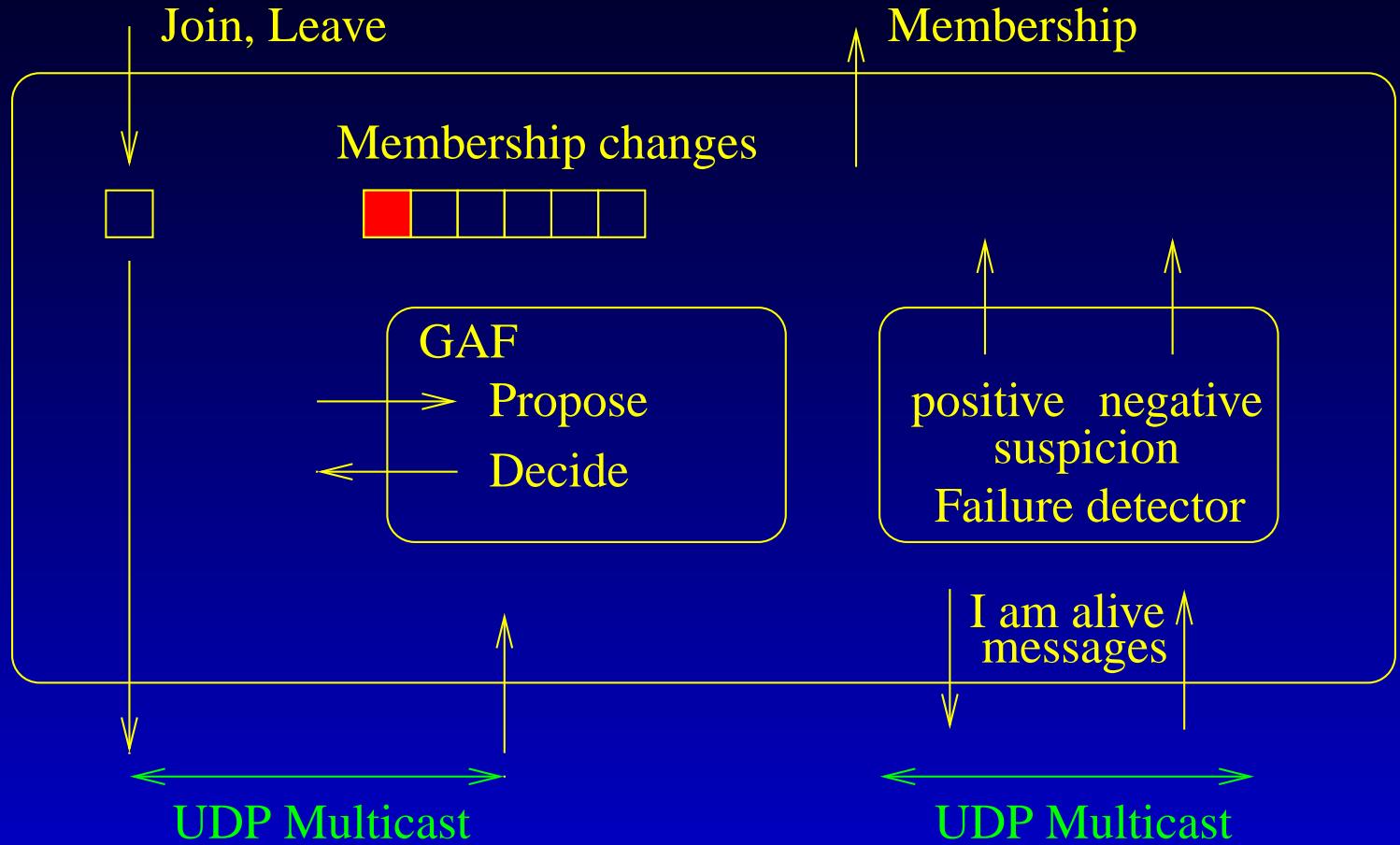
# Solving Membership



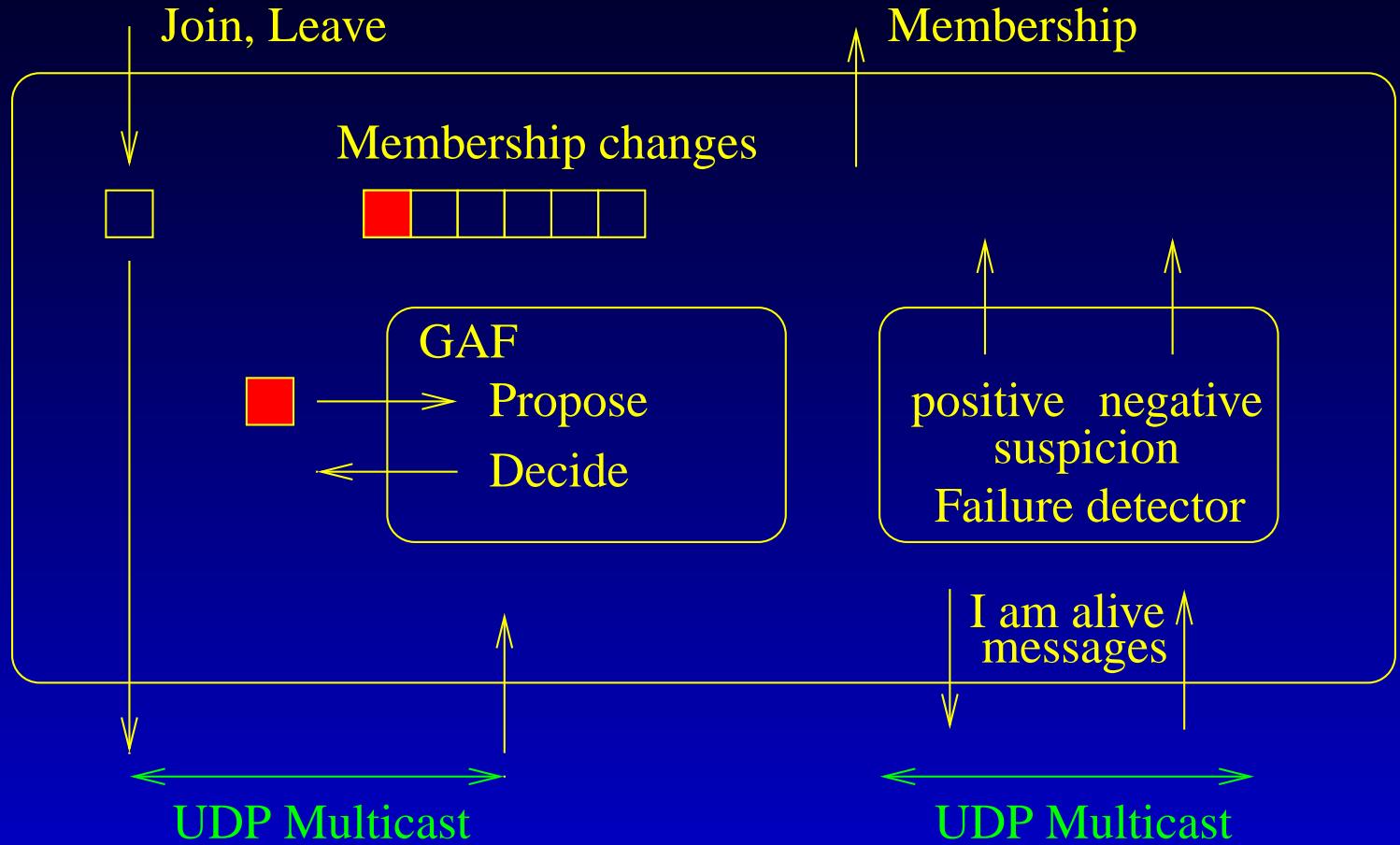
# Solving Membership



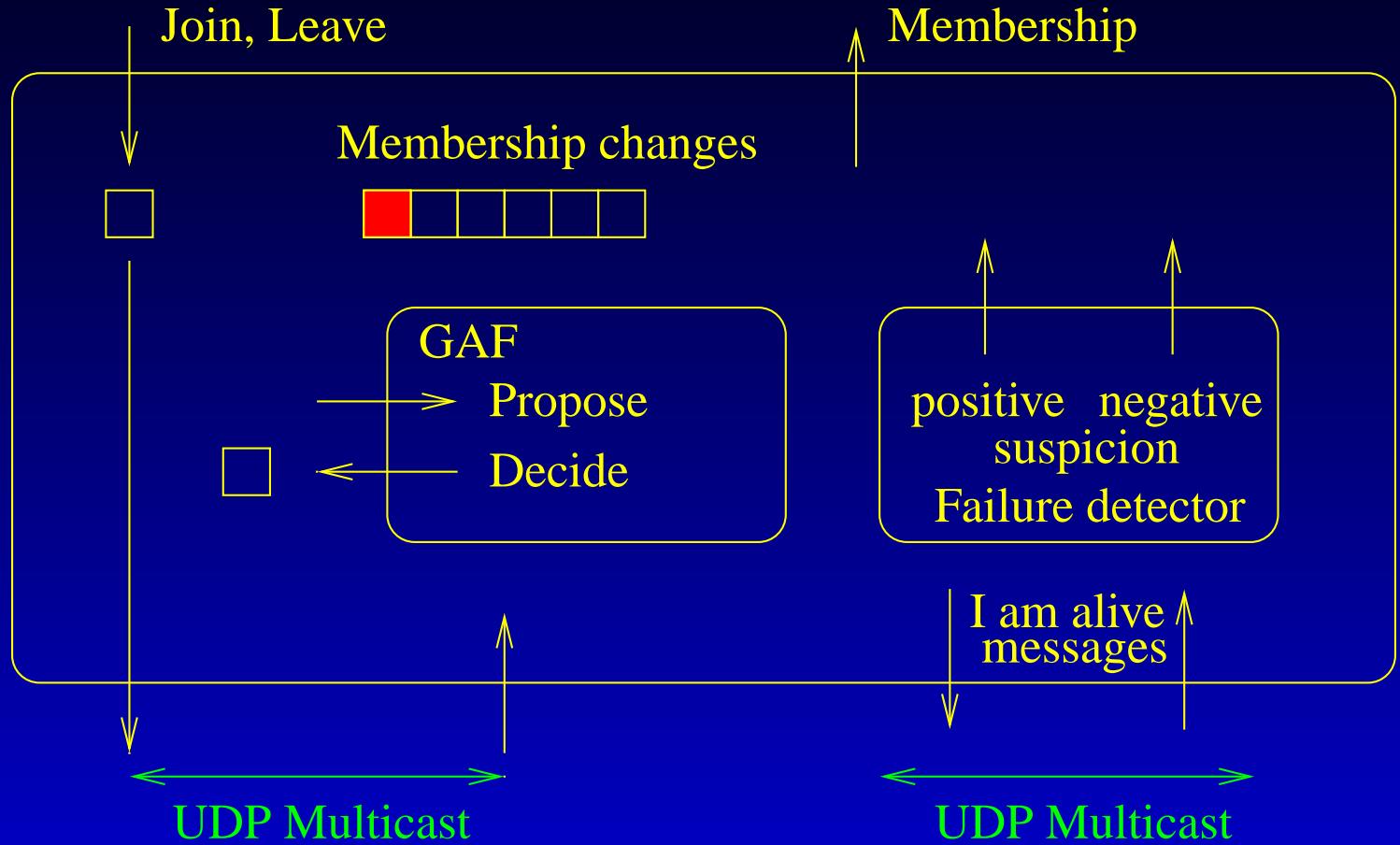
# Solving Membership



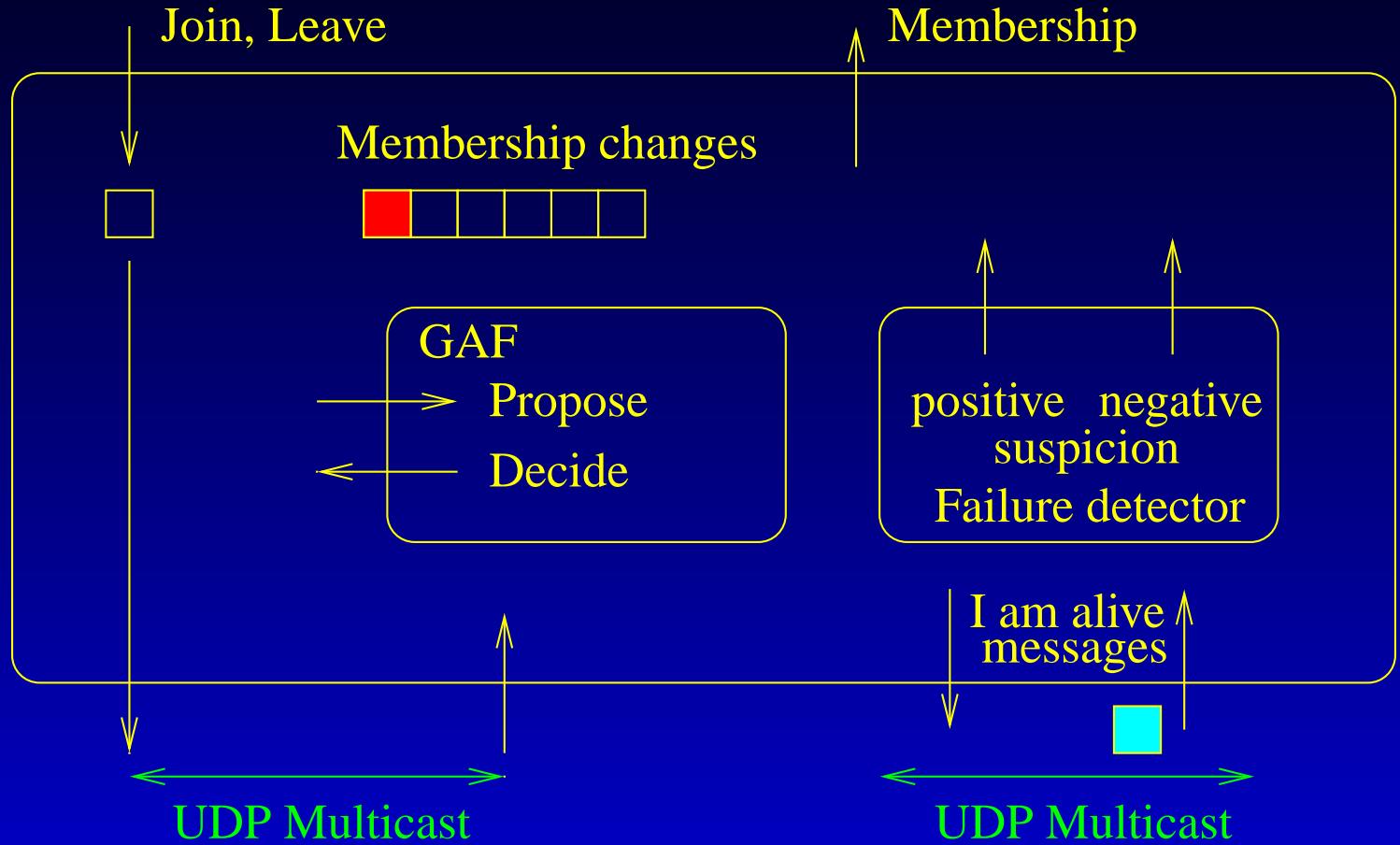
# Solving Membership



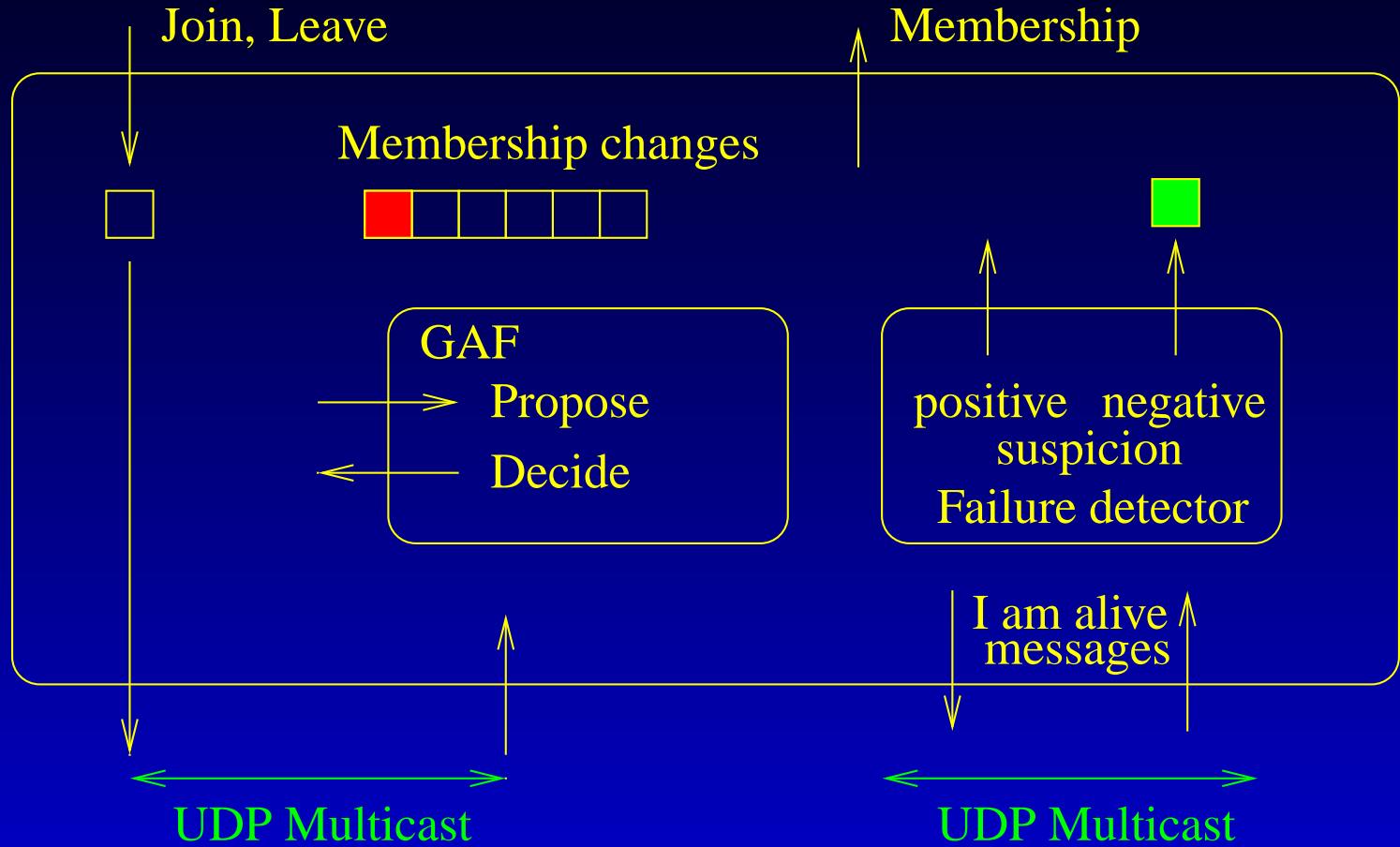
# Solving Membership



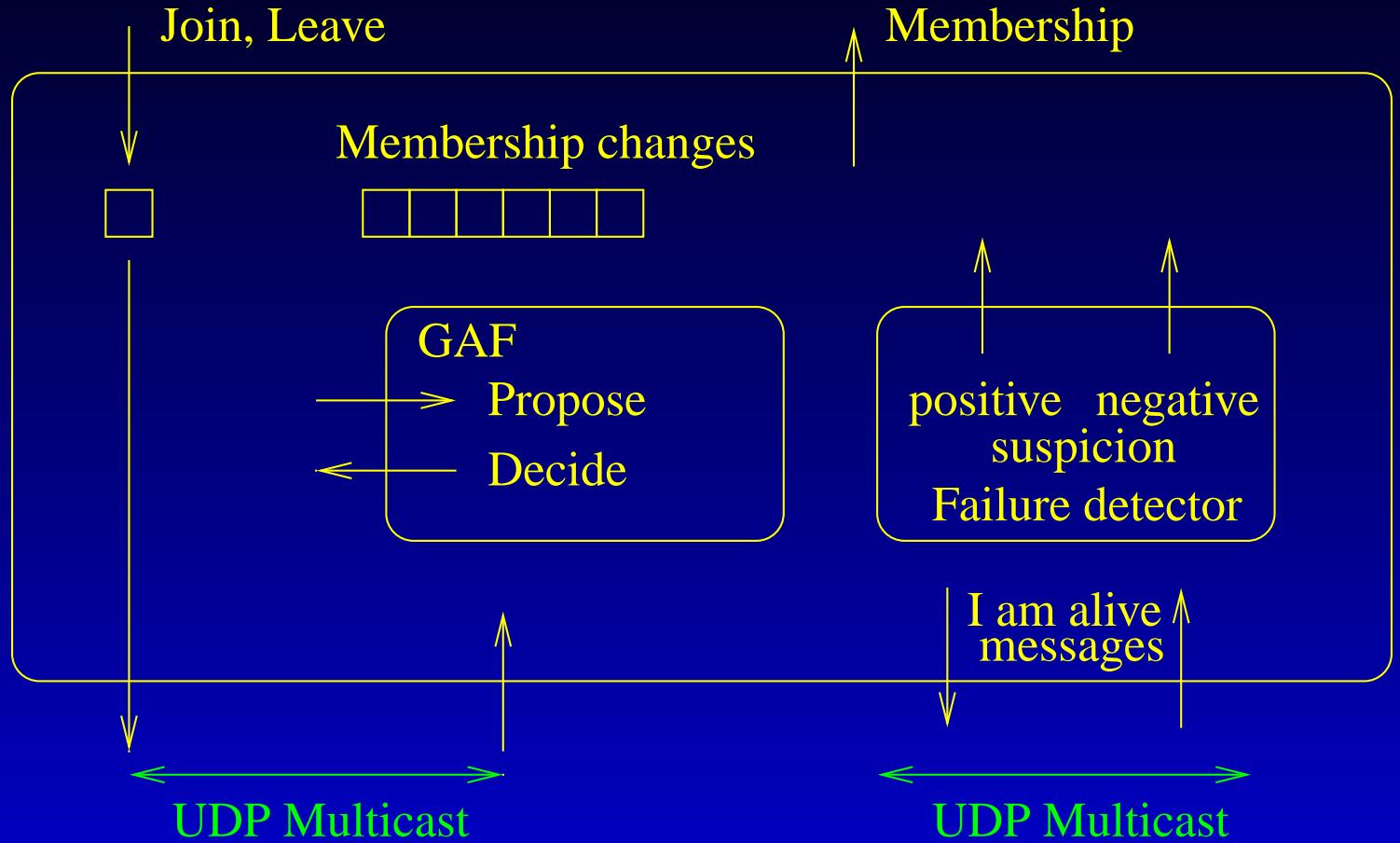
# Solving Membership



# Solving Membership



# Solving Membership



# Virtual Synchrony

# Virtual Synchrony

- Membership is a useful service, not only because it helps tracking the group composition changes, but also because it helps garbagging old broadcast messages that were still stored for possible future retransmissions.

# Virtual Synchrony

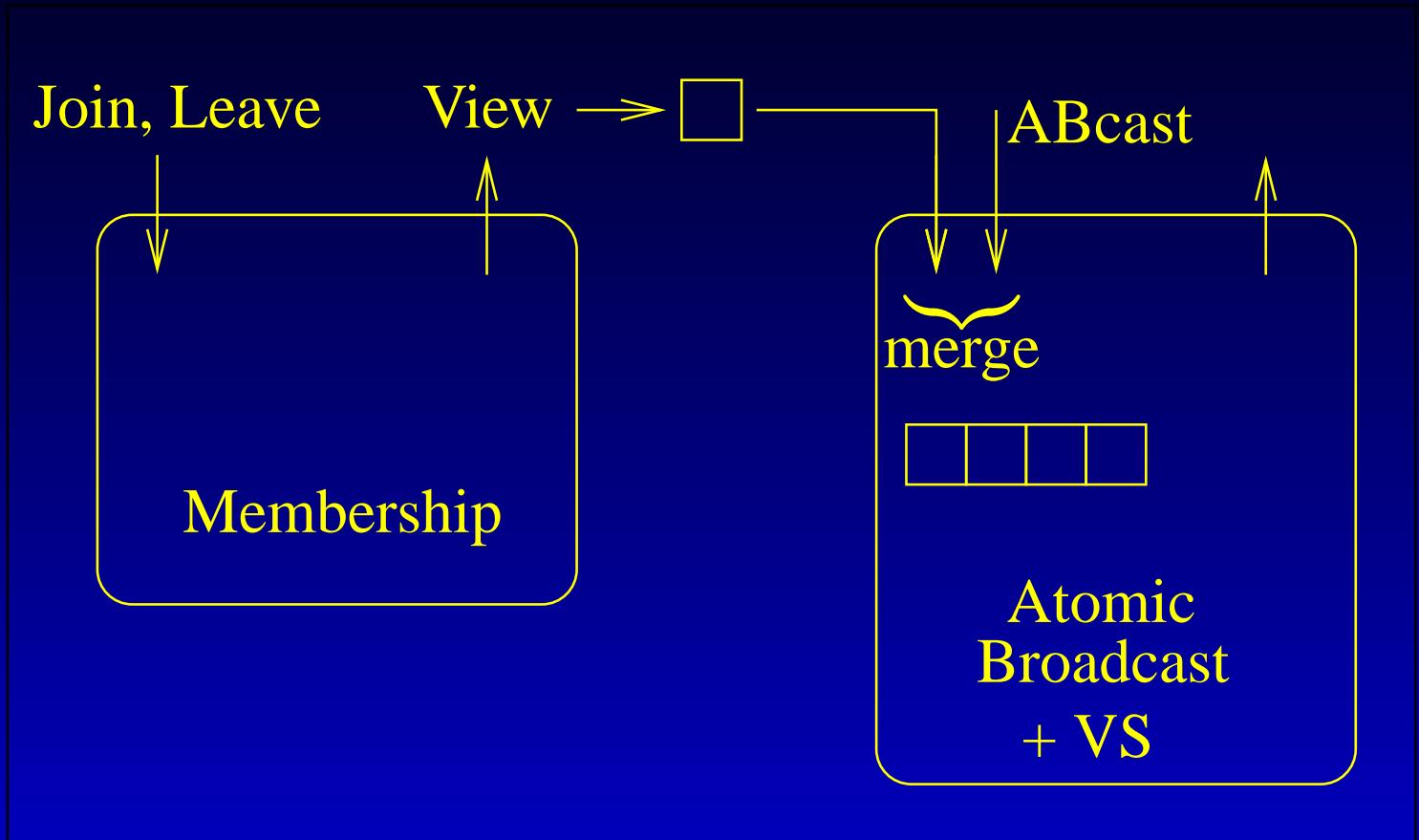
- Membership is a useful service, not only because it helps tracking the group composition changes, but also because it helps garbagging old broadcast messages that were still stored for possible future retransmissions.
- However, we also need to synchronize atomic broadcast with membership, to determine each time a view change occurs after which message (atomically broadcast) will this change be installed.

# Virtual Synchrony

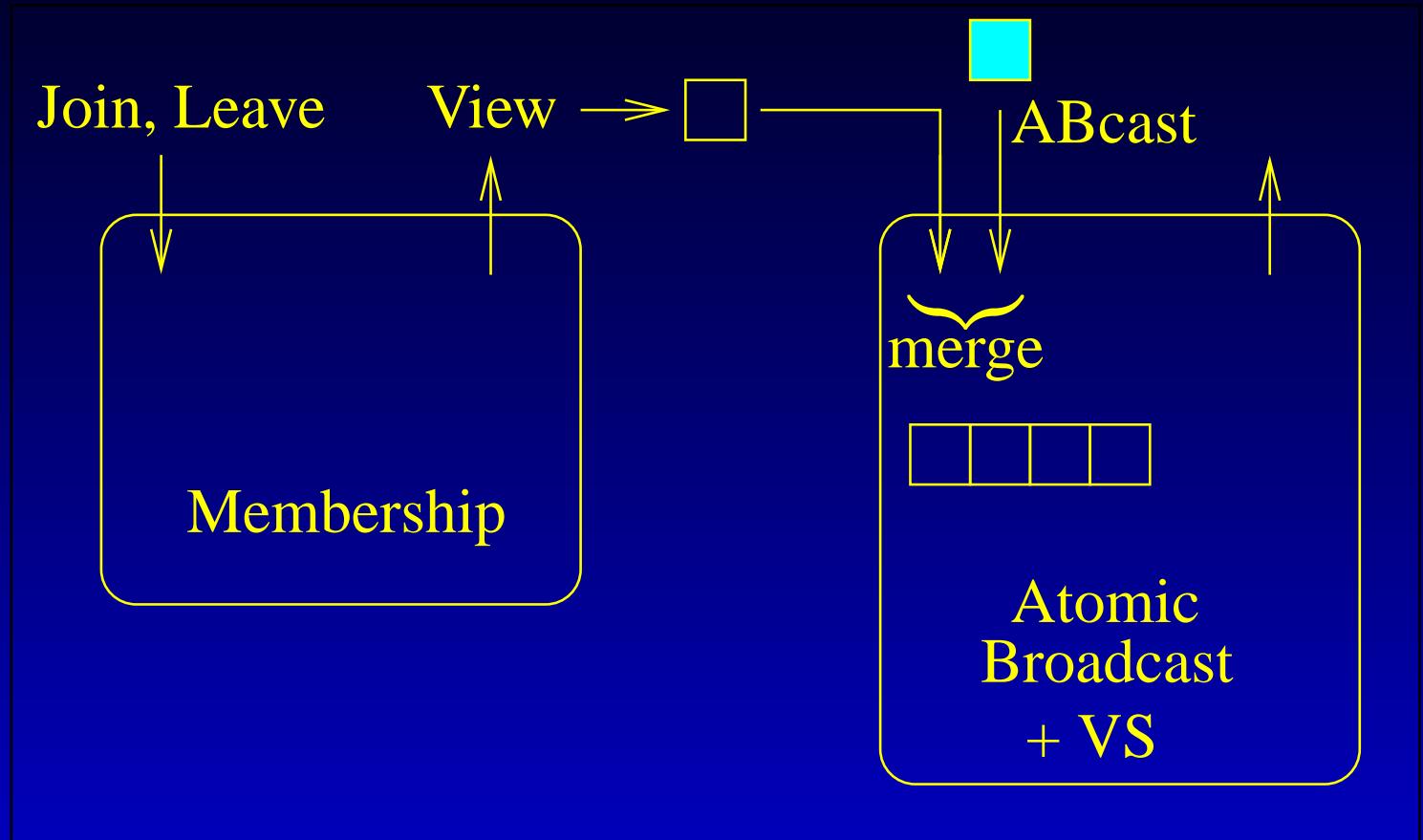
- Membership is a useful service, not only because it helps tracking the group composition changes, but also because it helps garbagging old broadcast messages that were still stored for possible future retransmissions.
- However, we also need to synchronize atomic broadcast with membership, to determine each time a view change occurs after which message (atomically broadcast) will this change be installed.
- This is the goal of Virtual Synchrony (VS).

# A Complex Solution to VS

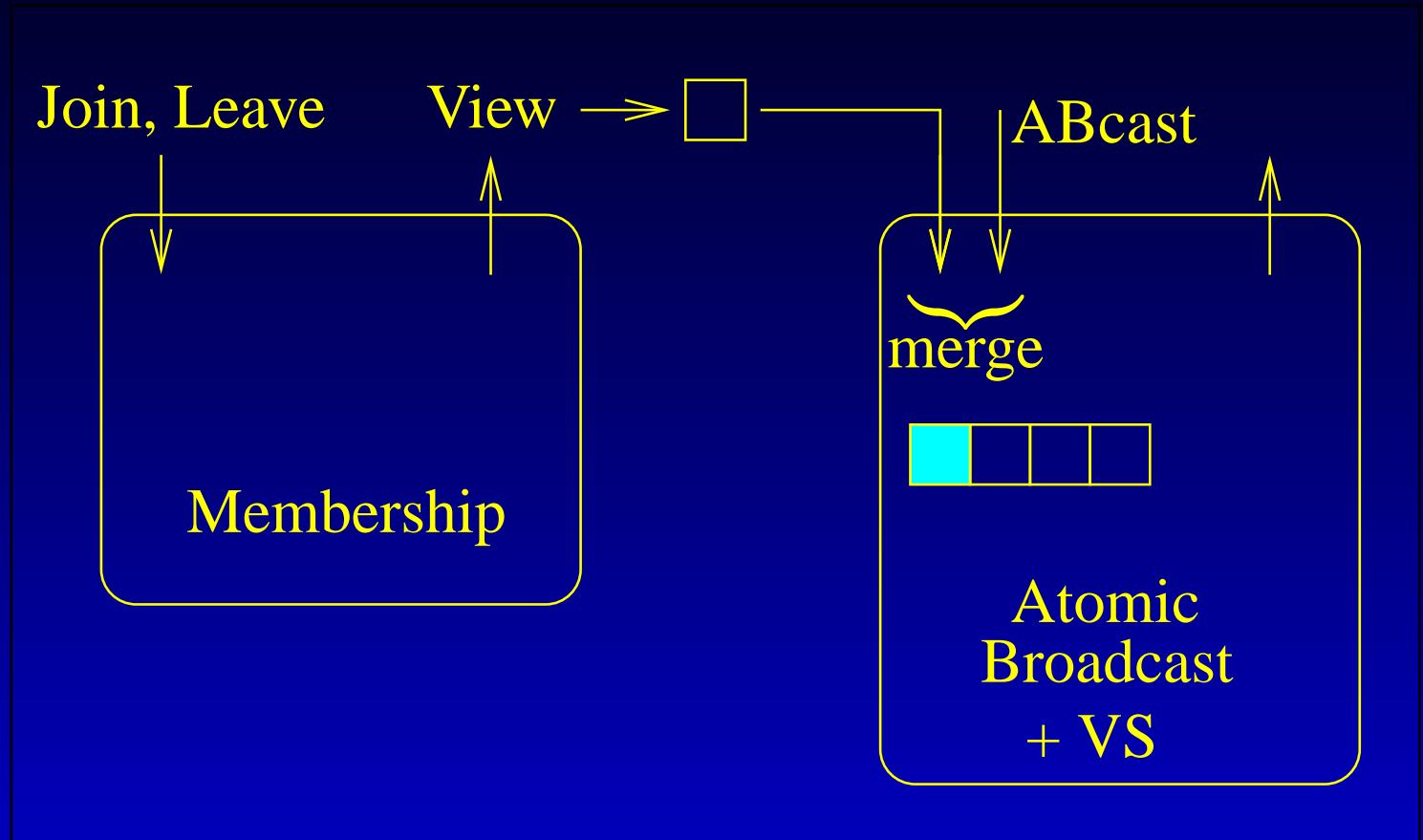
# A Complex Solution to VS



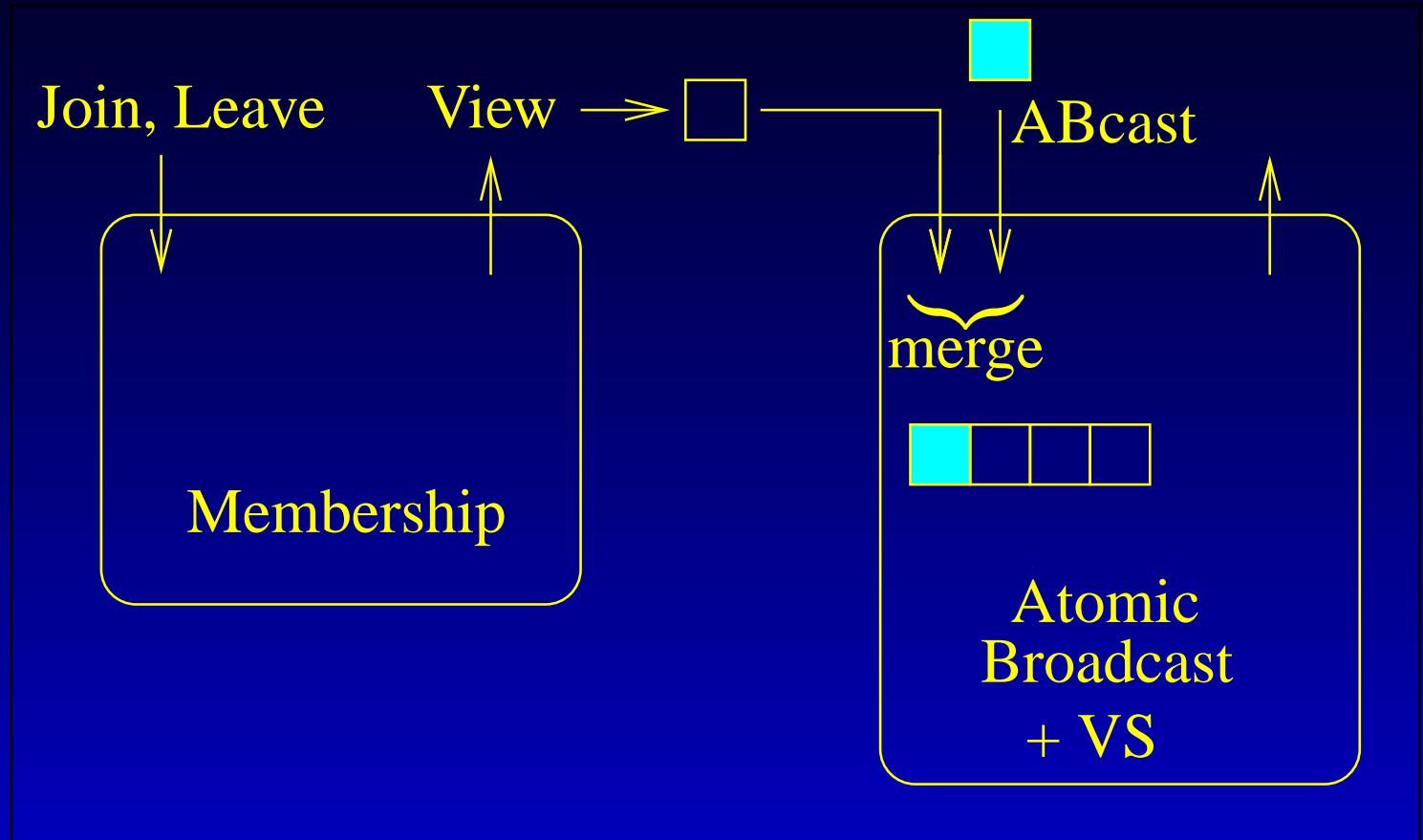
# A Complex Solution to VS



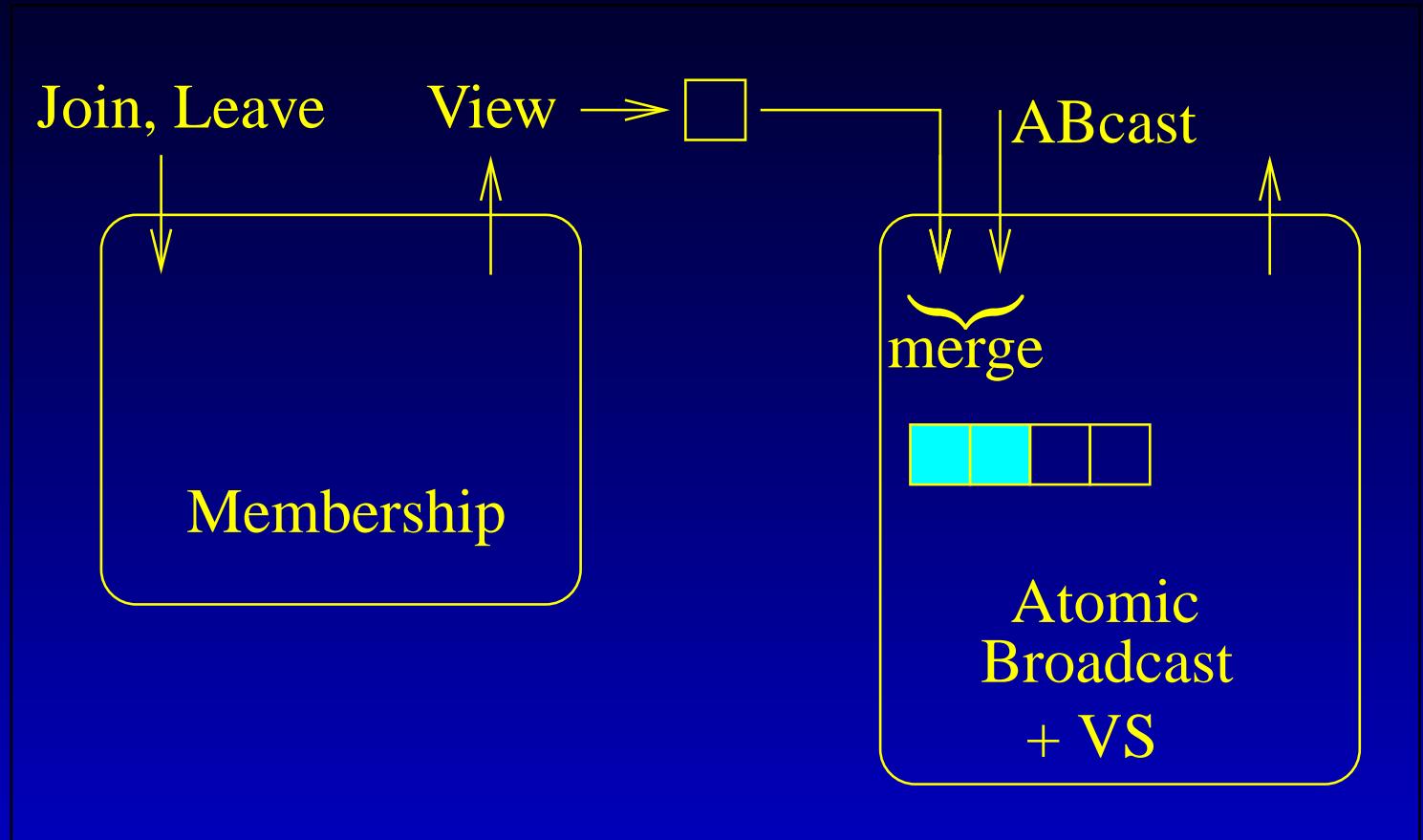
# A Complex Solution to VS



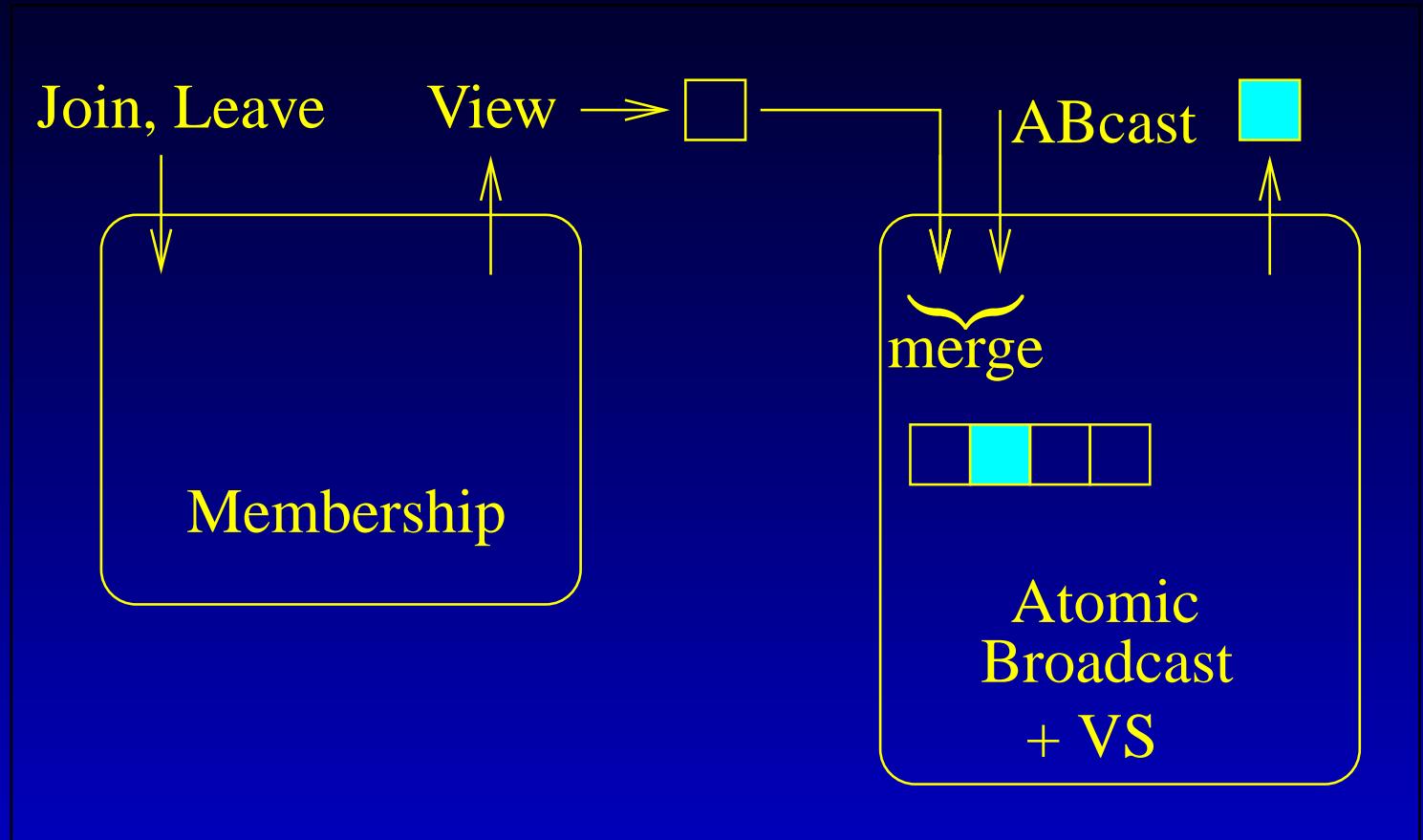
# A Complex Solution to VS



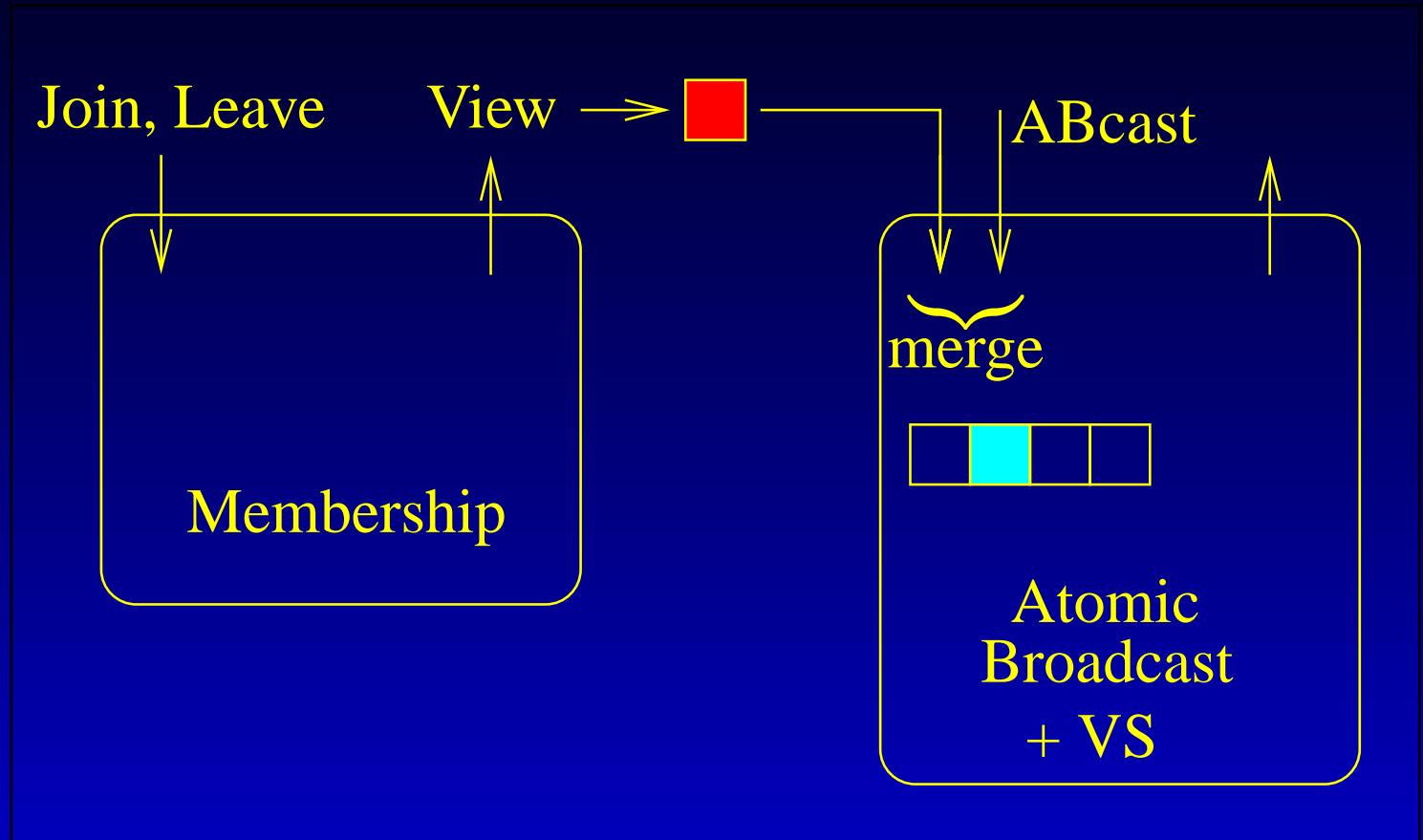
# A Complex Solution to VS



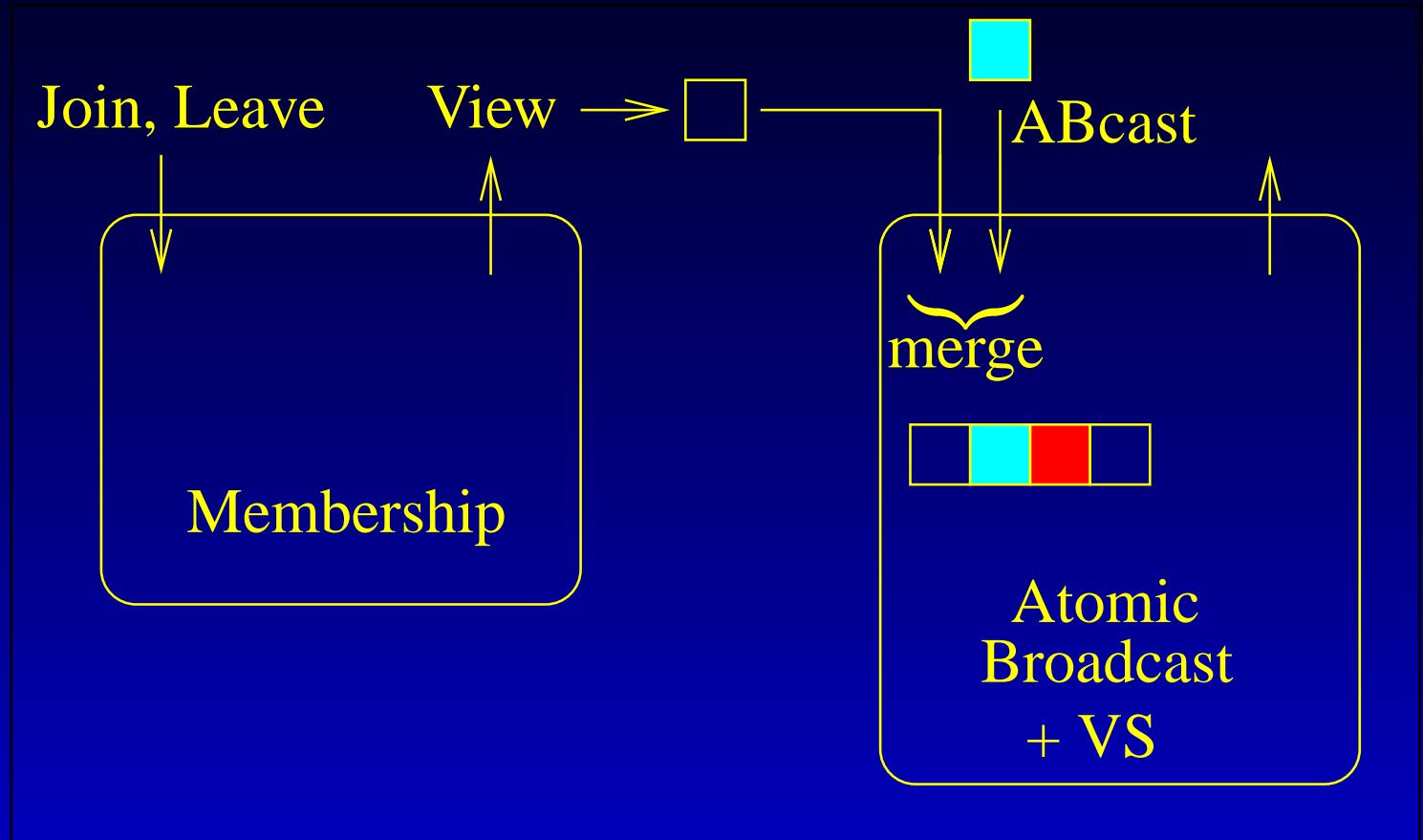
# A Complex Solution to VS



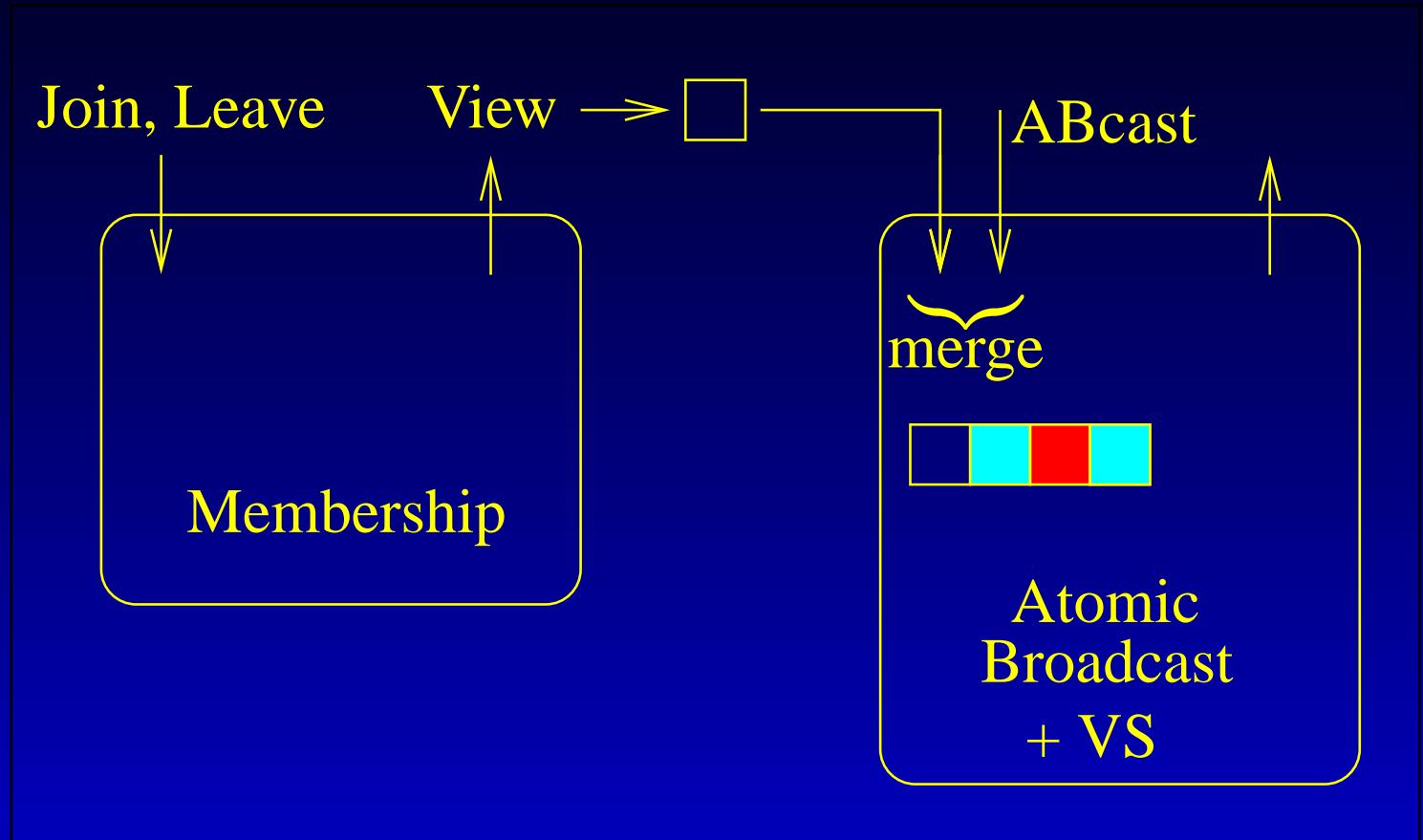
# A Complex Solution to VS



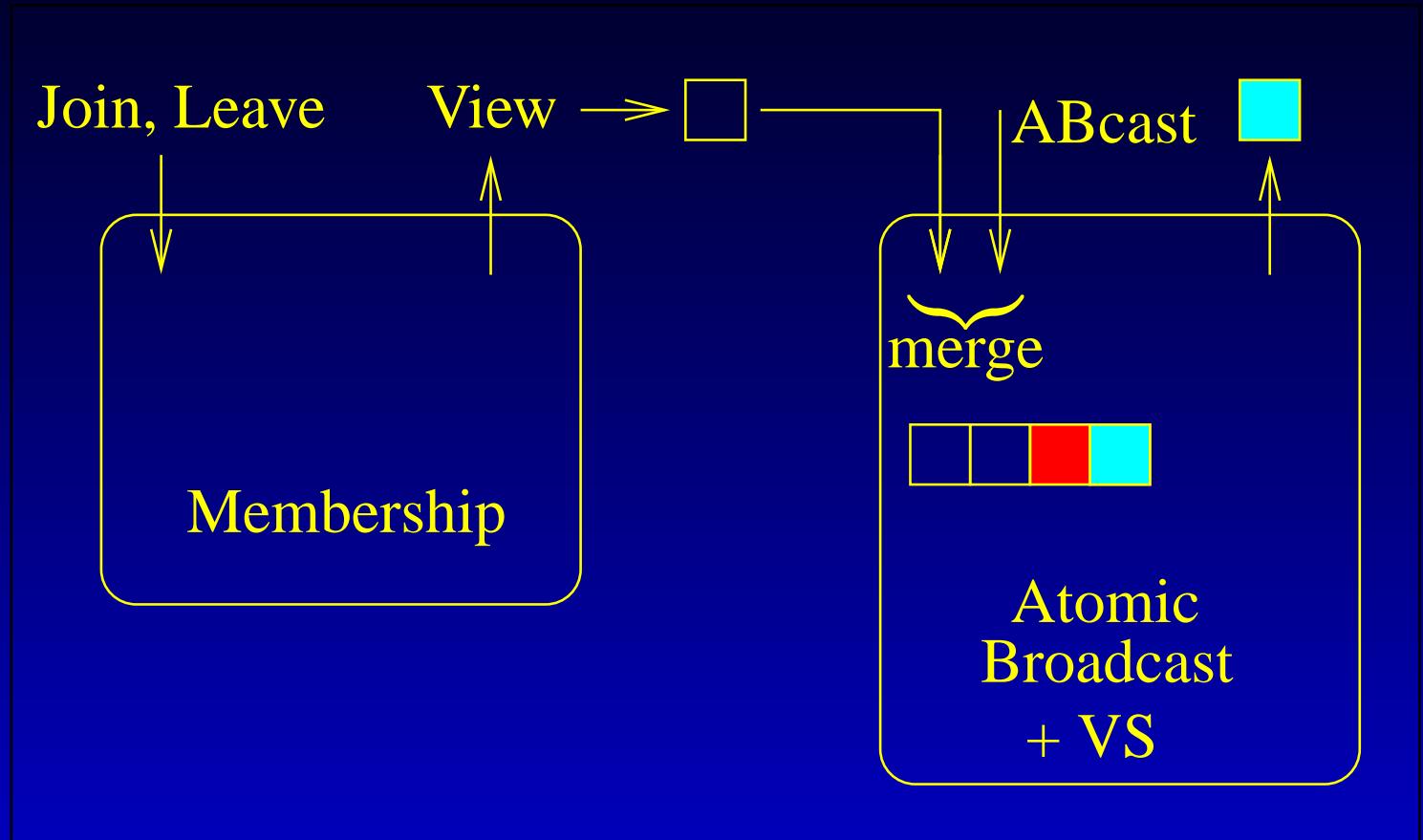
# A Complex Solution to VS



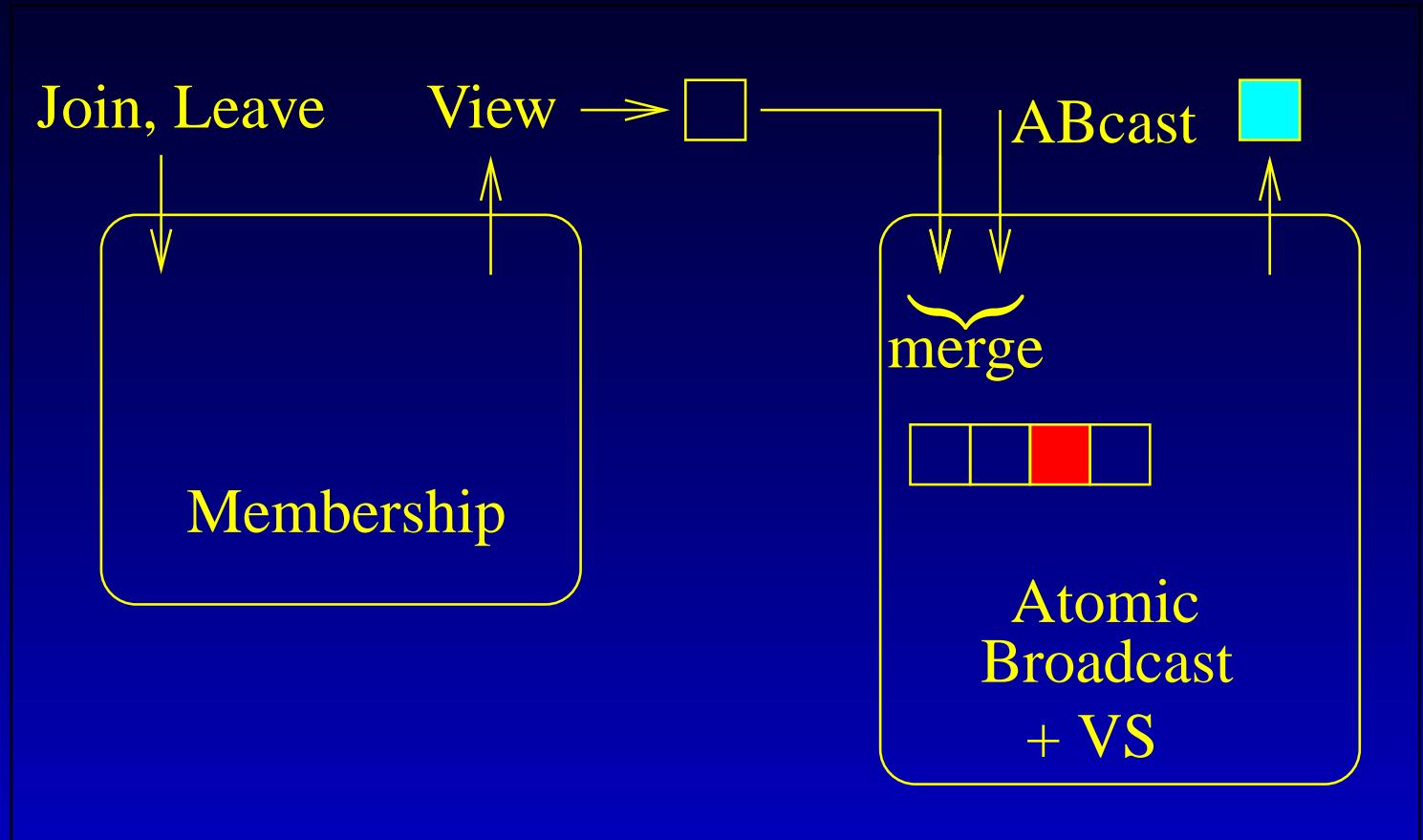
# A Complex Solution to VS



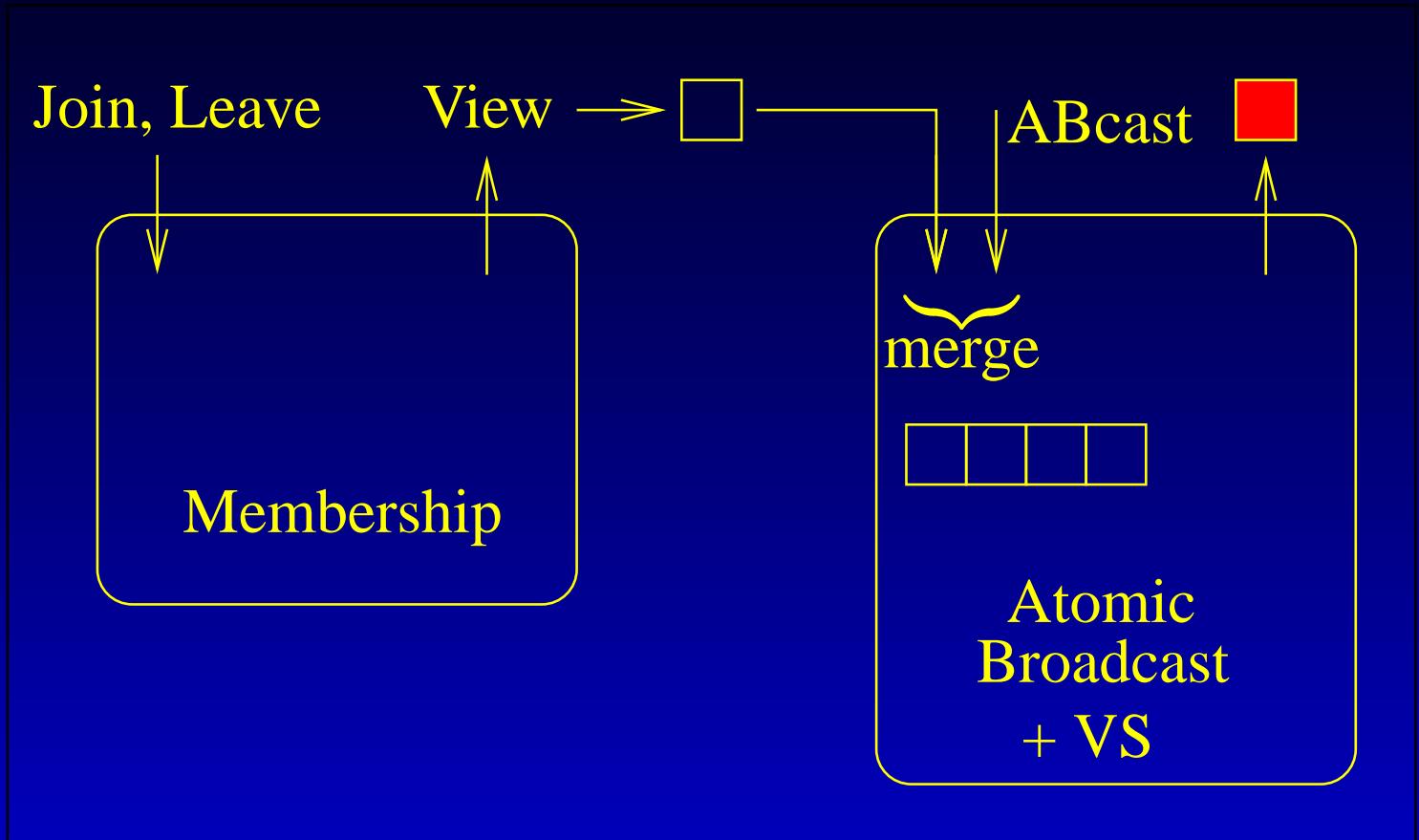
# A Complex Solution to VS



# A Complex Solution to VS



# A Complex Solution to VS



# Our Implementation of VS

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.
- That means that we have two agreement subdecisions for each decision taken by GAF :

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.
- That means that we have two agreement subdecisions for each decision taken by GAF :
  - A first subdecision for atomic broadcast;

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.
- That means that we have two agreement subdecisions for each decision taken by GAF :
  - A first subdecision for atomic broadcast;
  - A second subdecision for membership.

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.
- That means that we have two agreement subdecisions for each decision taken by GAF :
  - A first subdecision for atomic broadcast;
  - A second subdecision for membership.
- With this system, we do not need to instantiate as many GAF objects as agreement problems we have to solve

# Our Implementation of VS

- We mostly rely on the ability of GAF to decide for multiple agreement problems at the same time.
- That means that we have two agreement subdecisions for each decision taken by GAF :
  - A first subdecision for atomic broadcast;
  - A second subdecision for membership.
- With this system, we do not need to instantiate as many GAF objects as agreement problems we have to solve
- This approach saves machine resources, and also naturally synchronize membership wrt atomic broadcast.

# Future Extensions

# Future Extensions

- GAF makes possible to dynamically add and remove agreement problems on which decisions are taken.

# Future Extensions

- GAF makes possible to dynamically add and remove agreement problems on which decisions are taken.
- This makes possible to have initially only one agreement field in decisions. This field would be used by Adam to agree on the order of insertion/removing of others agreement problems.

# Future Extensions

- GAF makes possible to dynamically add and remove agreement problems on which decisions are taken.
- This makes possible to have initially only one agreement field in decisions. This field would be used by Adam to agree on the order of insertion/removing of others agreement problems.
- In the current implementation, this is done statically in the code.

# Future Extensions

- GAF makes possible to dynamically add and remove agreement problems on which decisions are taken.
- This makes possible to have initially only one agreement field in decisions. This field would be used by Adam to agree on the order of insertion/removing of others agreement problems.
- In the current implementation, this is done statically in the code. Also, it would be possible to have another agreement field, used to negotiate the QoS provided by GAF.

# Conclusion

# Conclusion

- Eden has been successfully interfaced to two different ORBs to provide a partial implementation of FT-Corba :

# Conclusion

- Eden has been successfully interfaced to two different ORBs to provide a partial implementation of FT-Corba :
  - Commercial ORB, orbacus

# Conclusion

- Eden has been successfully interfaced to two different ORBs to provide a partial implementation of FT-Corba :
  - Commercial ORB, orbacus
  - Research ORB, orbus, INRIA/South East University of Nanjing (China) cooperation.

# Conclusion

- Eden has been successfully interfaced to two different ORBs to provide a partial implementation of FT-Corba :
  - Commercial ORB, orbacus
  - Research ORB, orbus, INRIA/South East University of Nanjing (China) cooperation.
- One can also use Eden in a stand-alone fashion to create a group a replicated servers.