Hosted by Sourceforge since May 2000

INTRODUCTION TO JGROUPS

What is JGroups?

JGroups is a **reliable group communication toolkit** written entirely in **Java**.

	Unreliable	Reliable
Unicast	UDP	TCP
Multicast	IP Multicast	**JGroups**

It is based on **IP multicast** (although TCP can also be used as transport), but extends it with:

- **Reliability**.
- **Group membership**.

Features

Reliability includes (among other things):

- Lossless transmission of a message to all recipients;
- Fragmentation;
- Ordering of messages;
- Atomicity.

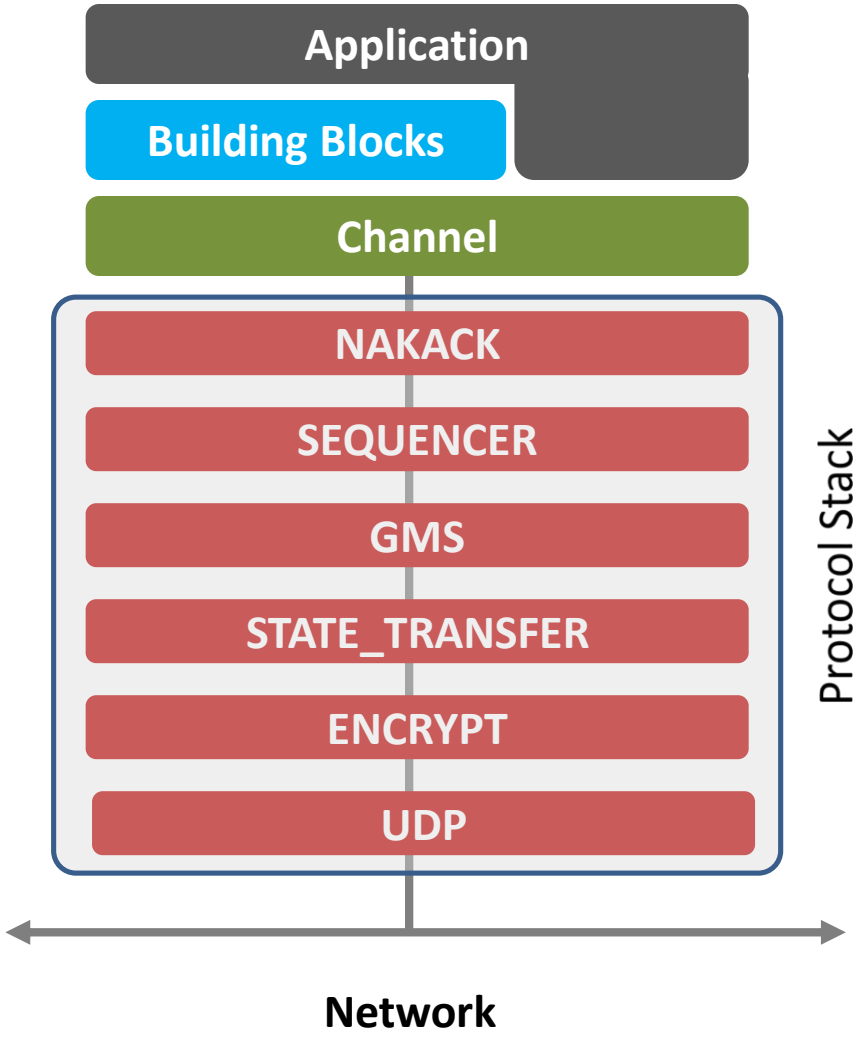
Group Membership includes:

- Knowledge of who the members of a group are;
- Membership change notification;

Why JGroups?

- Free open source license **Apache License 2.0** (since version 3.4)
- Easy to use and highly configurable.
- Good implementation using Java Programming Language with clear structure and well documentation.
- Extensible, save a lot of developer time when implement reliable messaging application and “write one run everywhere”.

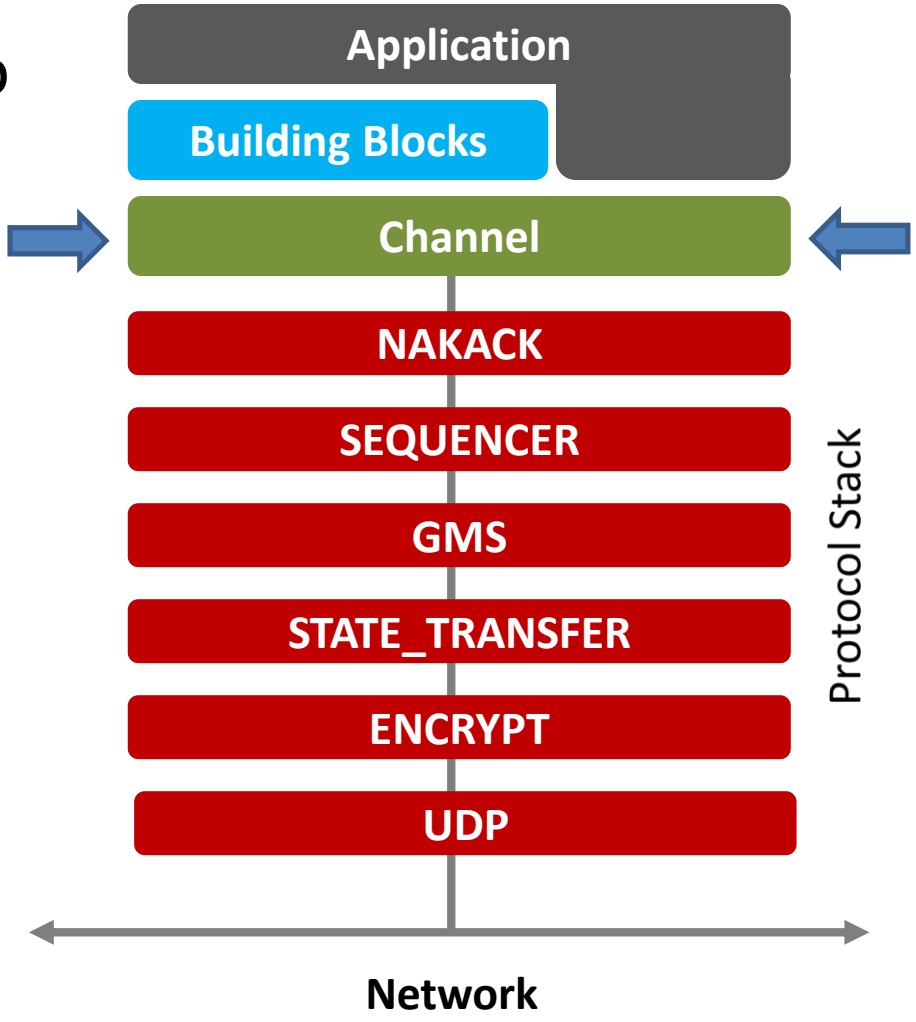
Architecture



CHANNEL

Channel

- Lower-Level Abstraction to build Multicast Communication
- Similar to **java.net.MulticastSocket**
  - Group membership
  - Reliability



Channel Operations

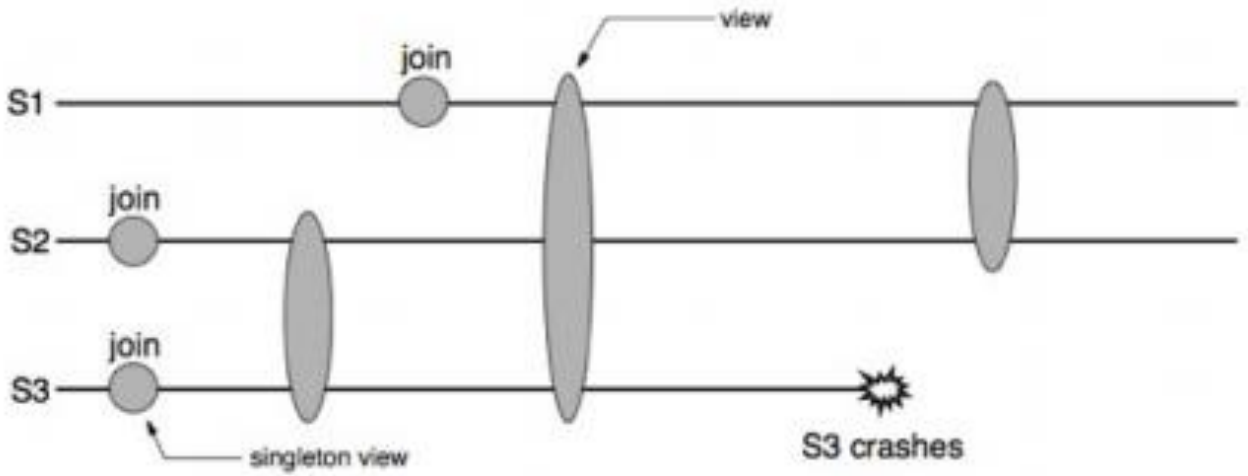
- Channel **creation**
- **Joining** a cluster
- **Send** a message
- **Receive** a message
- Retrieve **membership**
- Be **notified** when members **join**, **leave** (including **crashes**)
- **Disconnect** from the group
- **Close** the channel

Channel Operations - Example

```
JChannel channel=new JChannel("file://home/default.xml");
channel.setReceiver(new ReceiverAdapter(){
       public void viewAccepted(View new_view) {
              System.out.println("* view: " + new_view);
       }
       public void receive(Message msg) {
              System.out.println(msg.getSrc() + ": " +
                                          msg.getObject());
       }
});
channel.connect("demo-group");
System.out.println("members are: "+ channel.getView().getMembers());
Message msg=new Message(null, null, "Hello world");
channel.send(msg);
channel.disconnect();
channel.close();
```

View

- A **view** (org.jgroups.View) is a list of the current members of a group.
- It consists of a **ViewId**, which uniquely identifies the view, and a **list of members**.
- The first member of a view is the ***coordinator***
- Views are installed in a channel automatically whenever a new member joins or an existing one leaves (or crashes).



JGroups

Channel Operations - Creation

- A channel is created using one of its public constructors

```
public JChannel()
public JChannel(String props)
```

  - The props argument points to an XML file containing the configuration of the protocol stack to be used.

- Example:

```
JChannel channel=new JChannel("file://home/default.xml");
```

Channel Operations - Joining a cluster

- When a client wants to join a cluster, it *connects* to a channel giving the name of the cluster to be joined:

```
public void connect(String cluster)
```

- Example:

```
channel.connect("demo-group");
```

Message

- Data is sent between members in the form of messages (org.jgroups.Message)
- A message has 5 fields

  - **Destination address**: The address of the receiver. If null, the message will be sent to all current group members
  - **Source address**: The address of the sender. Can be left null, and will be filled in by the transport protocol
  - **Flags**: the currently recognized flags are OOB, DONT_BUNDLE, NO_FC, NO_RELIABILITY, NO_TOTAL_ORDER, NO_RELAY and RSVP
  - **Headers:** a list of headers that can be attached to a message
  - **Payload:** the actual data

Channel Operations - send

- Once the channel is connected, messages can be sent using one of the send() methods:

```
public void send(Message msg)
public void send(Address dst, Serializable obj)
public void send(Address dst, Address src, Serializable
obj)
```

  - The message's destination should either be the address of the receiver (unicast) or null (multicast)

- Example:

```
Message msg = new Message(null, null, "Hello world");
channel.send(msg);
```

Channel Operations – Receive

- Method **receive**() in **ReceiverAdapter** (or Receiver) can be overridden to receive messages

```
public void receive(Message msg)
```

- **Example:**

```
channel.setReceiver(new ReceiverAdapter(){
                //…
                public void receive(Message msg) {
                System.out.println(msg.getSrc() + ": " +
                                                msg.getObject());
        }
});
```

Channel Operations – Receiving view changes

- The **viewAccepted()** callback of **ReceiverAdapter** can be used to get callbacks whenever a cluster membership change occurs.

- **Example**

```
channel.setReceiver(new ReceiverAdapter(){
        public void viewAccepted(View new_view) {
                System.out.println("* view: " + new_view);
        }
        //..
});
```

Channel Operations – Disconnect

- Disconnecting from a channel is done using the following method:

```
public void disconnect();
```

  - After a successful disconnect, the channel will be in the unconnected state, and may subsequently be reconnected.

- **Example:**

```
channel.disconnect();
```

Channel Operations – Close

- To destroy a channel instance (destroy the associated protocol stack, and release all resources), method **close**() is used:

```
public void close();
```

  - Closing a connected channel disconnects the channel first
- **Example:**

```
channel.close();
```

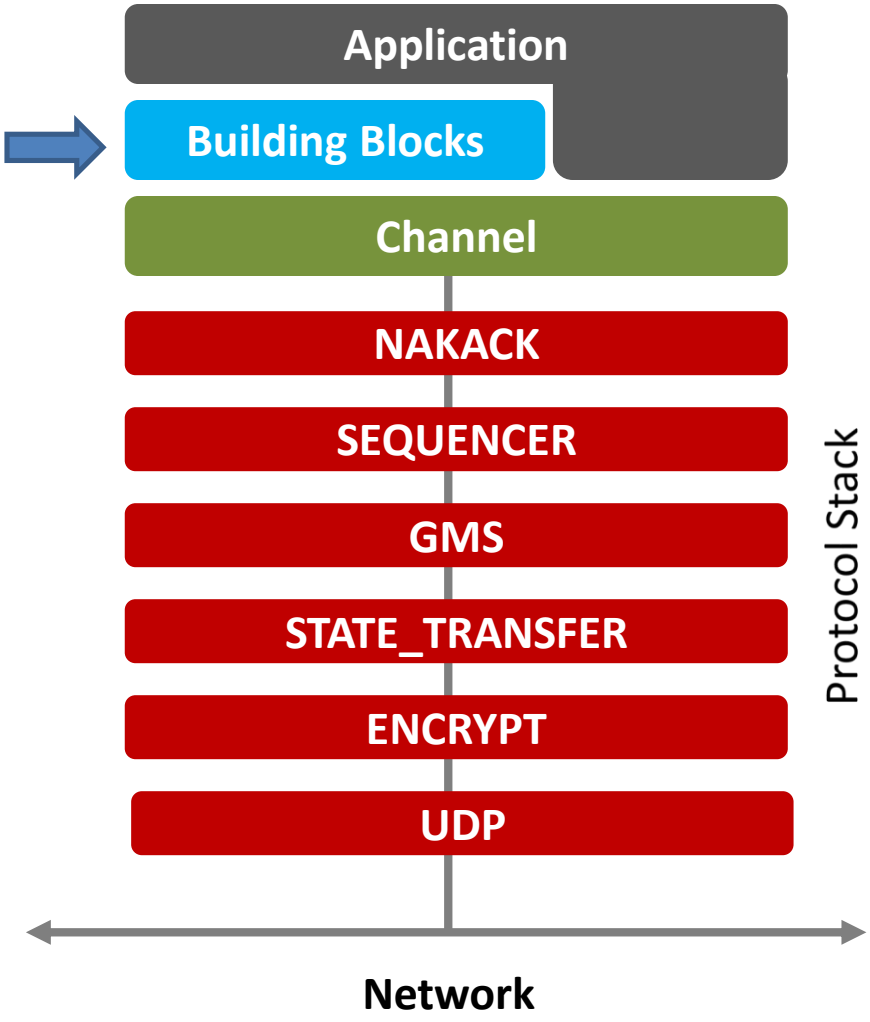
Channel Operations - Example

```
JChannel channel=new JChannel("file://home/default.xml");
channel.setReceiver(new ReceiverAdapter(){
        public void viewAccepted(View new_view) {
                System.out.println("* view: " + new_view);
        }
        public void receive(Message msg) {
                System.out.println(msg.getSrc() + ": " +
                                                msg.getObject());
        }
});
channel.connect("demo-group");
System.out.println("members are: "+ channel.getView().getMembers());
Message msg=new Message(null, null, "Hello world");
channel.send(msg);
channel.disconnect();
channel.close();
```

BUILDING BLOCKS

Building Blocks

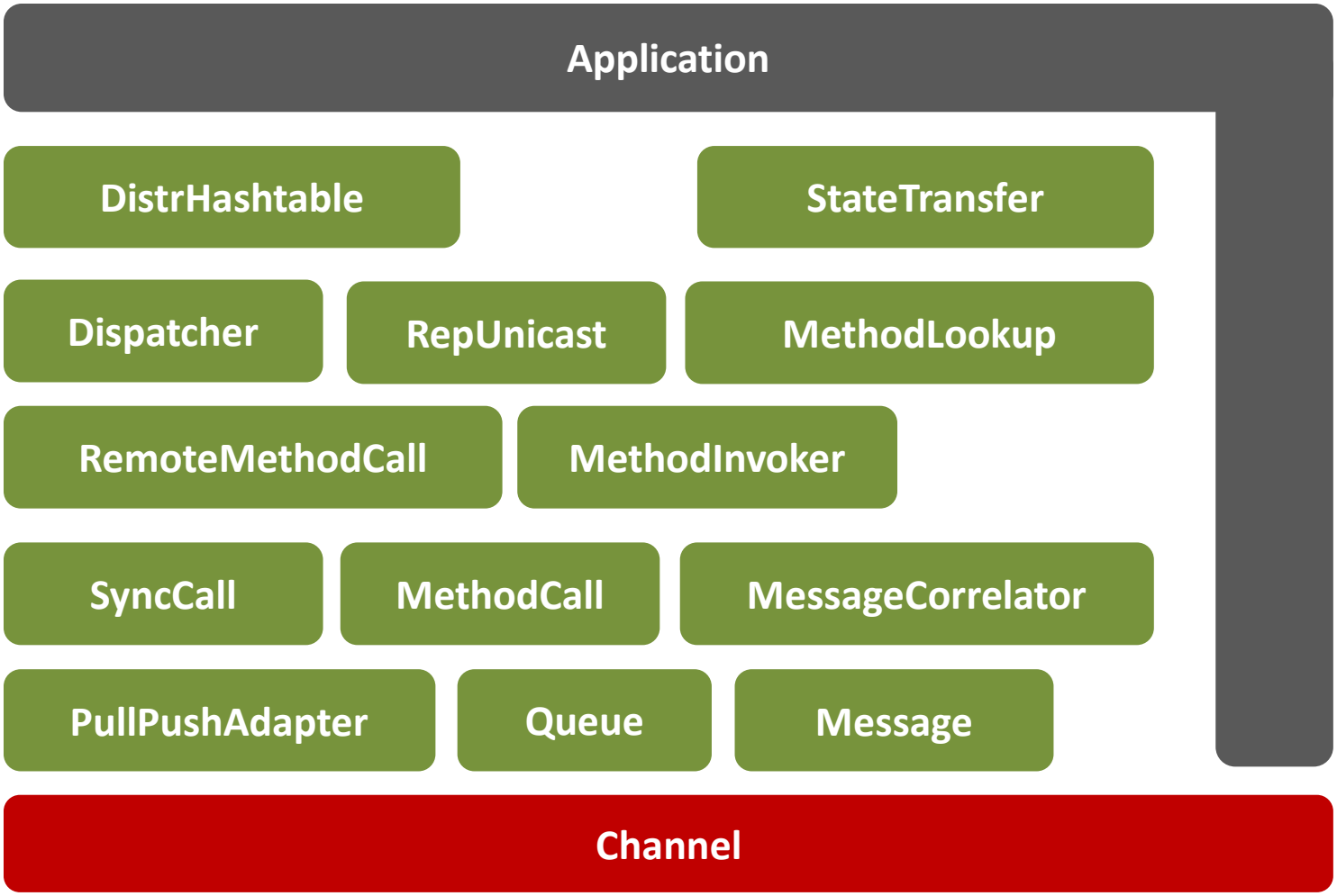
Building blocks are layered on **top of channels**, and can be used instead of channels whenever a **higher-level interface** is required.



Building Blocks

- Group communication pattern;
- Layered on **top of channels**;
- More **sophisticated** APIs;
- Applications **communicate directly** with the building block, rather than the channel;

Building Blocks



PROTOCOL STACK

Flexible Protocol Stack

Consider an example: