Babel: A framework for developing performant and dependable distributed protocols

Pedro Fouto, Pedro Ákos Costa,
Nuno Preguiça, João Leitão

SRDS 2022

Motivation

- Distributed protocol designs are often **simple** (in pseudocode), however implementing them is **error-prone and time consuming**, with lots of low level aspects, such as:
  - Networking
  - Concurrency
  - Multiplexing
  - Protocol interactions
  - …

Motivation

- We require a framework that **simplifies the development of distributed systems**:
  - **Abstracting low level aspects**: network, concurrency, timers, etc.
  - Without restricting **generability**, allowing implementing any distributed protocol.
  - **Performant** and allowing the implementation of **production-ready** systems.

Babel: Overview

- Java framework providing an **event driven programming model.**
- **Simplifies implementation** of distributed algorithms by trivially translating their specification (pseudo-code).
- Provided abstractions are sufficiently **generic** to support a wide variety of algorithms.
- Capable of supporting **production-ready performant** implementations.

Babel: Overview

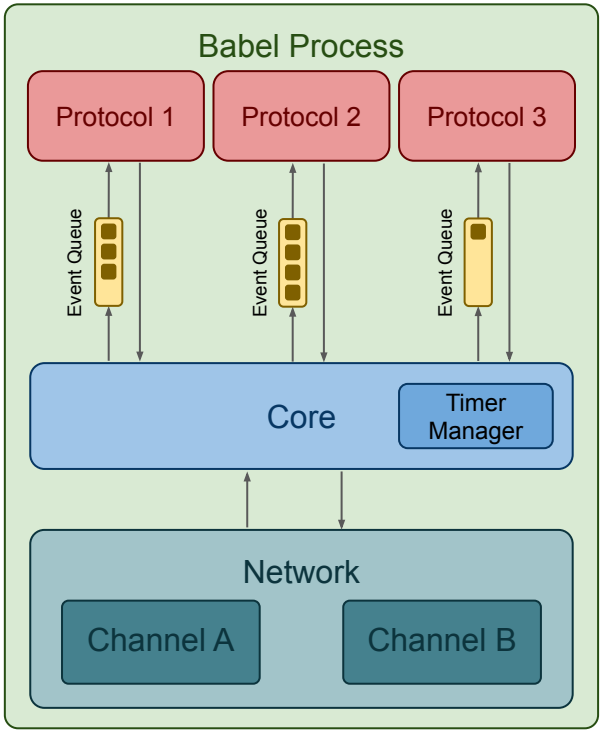
- Java framework providing an **e**
- **Simplifies implementation** of
- Provided abstractions are suffi
- Capable of supporting **produc**

g their

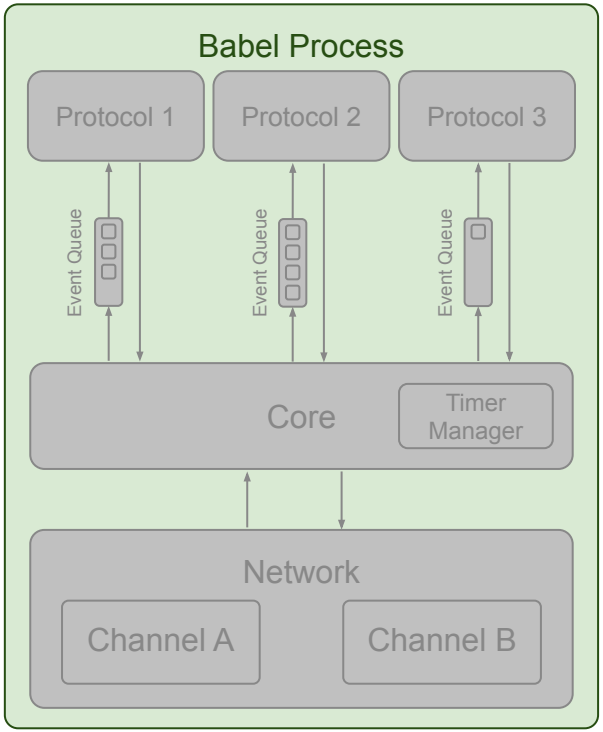
algorithms.

```
upon init do
    Send(JOIN, contactNode, myself);

upon Receive(JOIN, newNode) do
    trigger addNodeActiveView(newNode)
    foreach n ∈ activeView and n ≠ newNode do
        Send(FORWARDJOIN, n, newNode, ARWL, myself)

upon Receive(FORWARDJOIN, newNode, timeToLive, sender) do
    if timeToLive== 0||#activeView== 1 then
        trigger addNodeActiveView(newNode)
    else
        if timeToLive==PRWL then
            trigger addNodePassiveView(newNode)
        n ⟵ n ∈ activeView and n ≠ sender
        Send(FORWARDJOIN, n, newNode, timeToLive-1, myself)

upon dropRandomElementFromActiveView do
    n ⟵ n ∈ activeView
    Send(DISCONNECT, n, myself)
    activeView ⟵ activeView \{n}
    passiveView ⟵ passiveView ∪{n}

upon addNodeActiveView(node) do
    if node ≠ myself and node ∉ activeView then
        if isfull(activeView) then
            trigger dropRandomElementFromActiveView
        activeView ⟵ activeView ∪ node

upon addNodePassiveView(node) do
    if node ≠ myself and node ∉ activeView and node ∉ passiveView then
        if isfull(passiveView) then
            n ⟵ n ∈ passiveView
            passiveView ⟵ passiveView \{n}
        passiveView ⟵ passiveView ∪ node

upon Receive(DISCONNECT, peer) do
    if peer ∈ activeView then
        activeView ⟵ activeView \ {peer}
        addNodePassiveView(peer)
```

Babel: Architecture

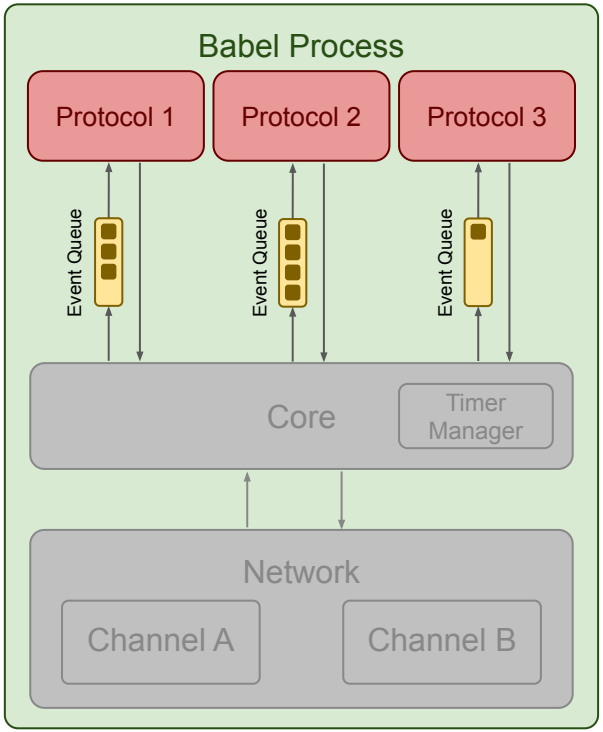


Babel: Architecture



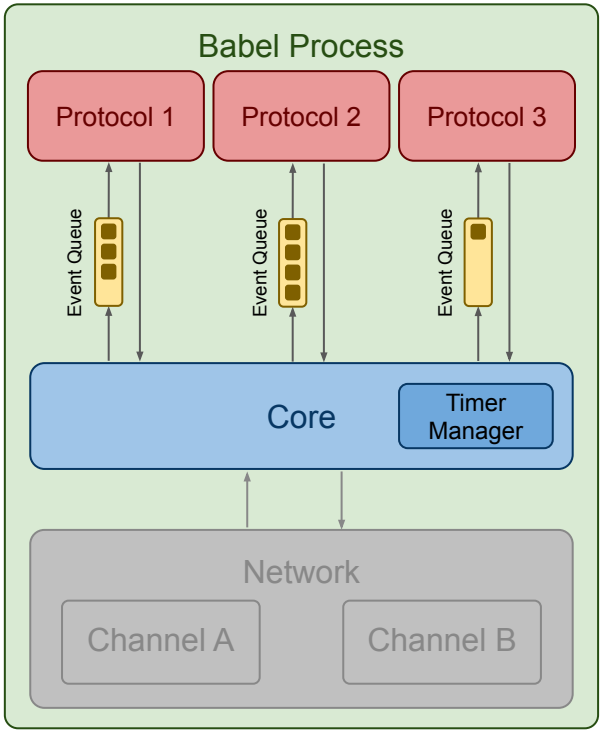
- Each *Babel process* represents a process of the distributed algorithm.

Babel: Architecture



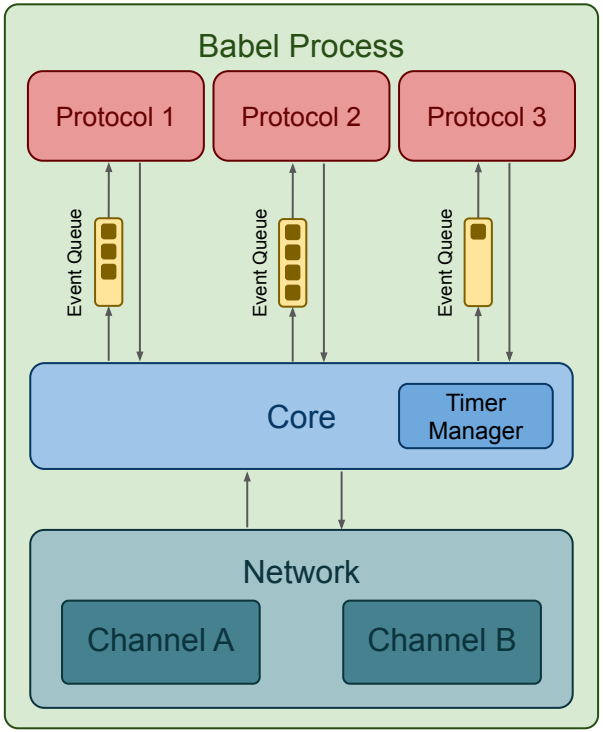
- Each *Babel process* represents a process of the distributed algorithm.
- *Protocols* are implemented by developers and encode the behavior of the algorithm.

Babel: Architecture



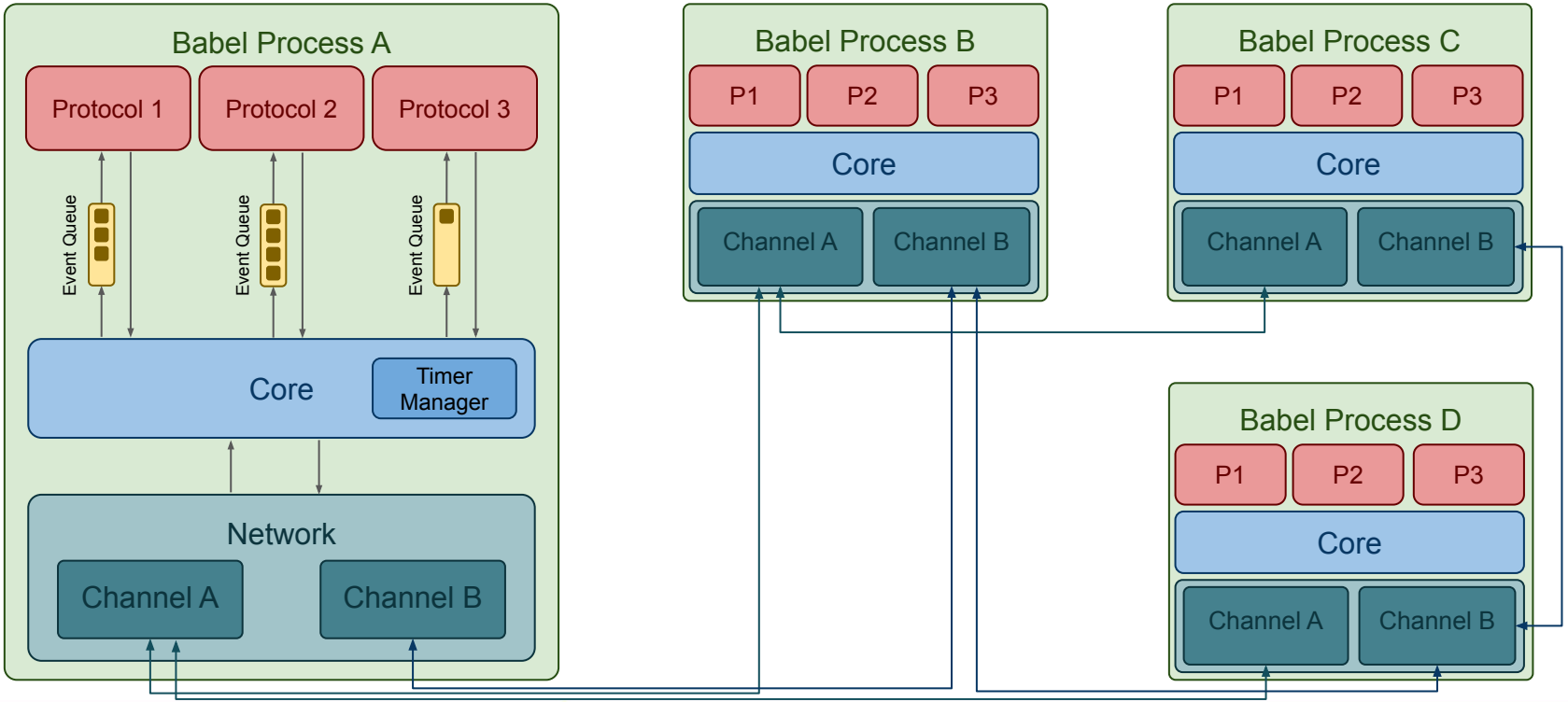
- Each *Babel process* represents a process of the distributed algorithm.
- *Protocols* are implemented by developers and encode the behavior of the algorithm.
- The *Core* coordinates and mediates interactions between protocols and processes.

Babel: Architecture

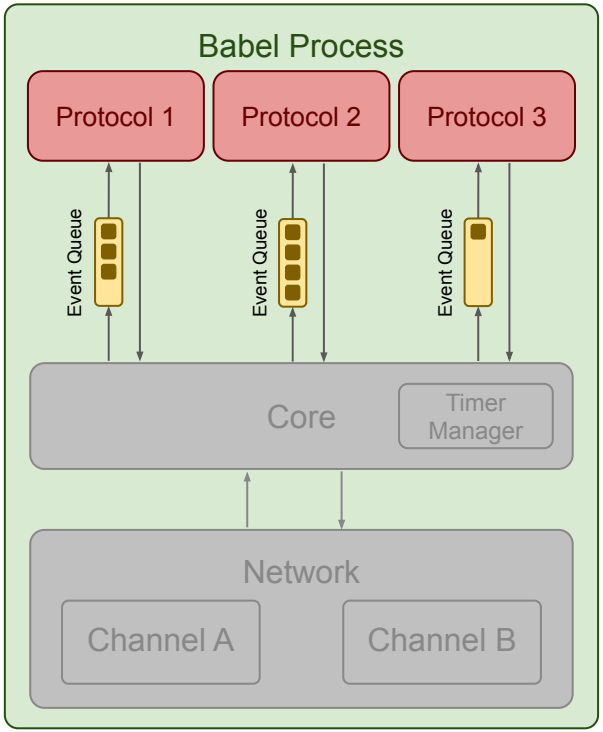


- Each *Babel process* represents a process of the distributed algorithm.
- *Protocols* are implemented by developers and encode the behavior of the algorithm.
- The *Core* coordinates and mediates interactions between protocols and processes.
- *Network channels* abstract communications between processes and provide different behaviours and guarantees.

Babel: Architecture



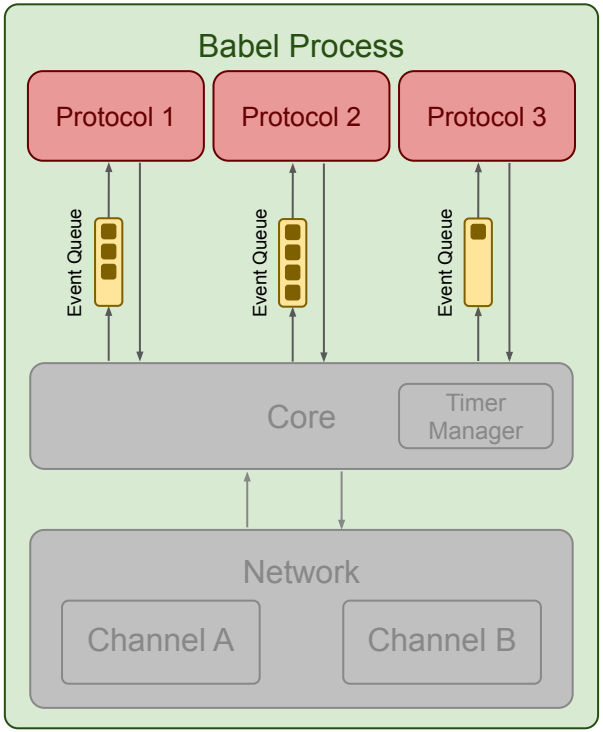
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.

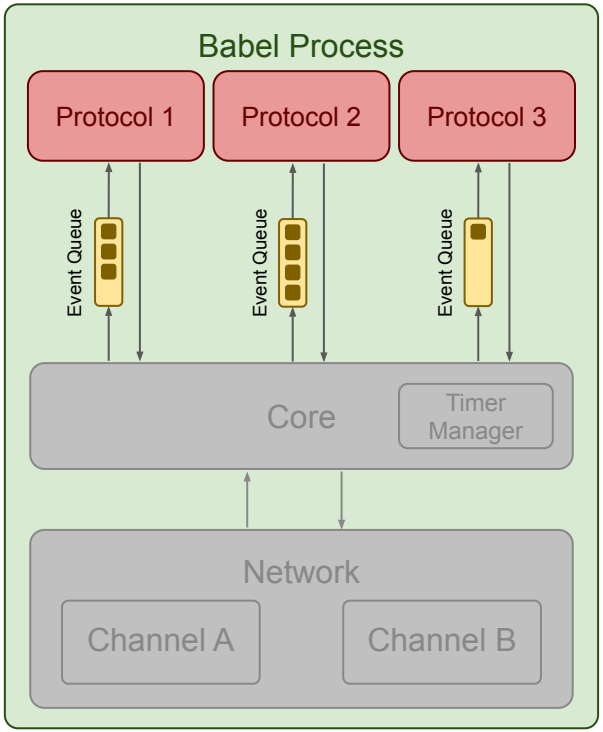
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.
- Received events are added to the event queue.

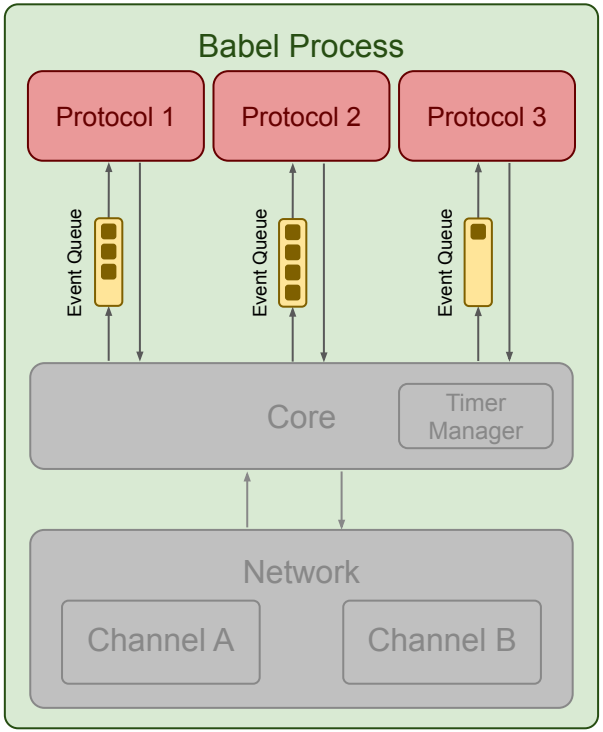
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.
- Received events are added to the event queue.
- **Dedicated thread** per protocol handles events in **serial** fashion.

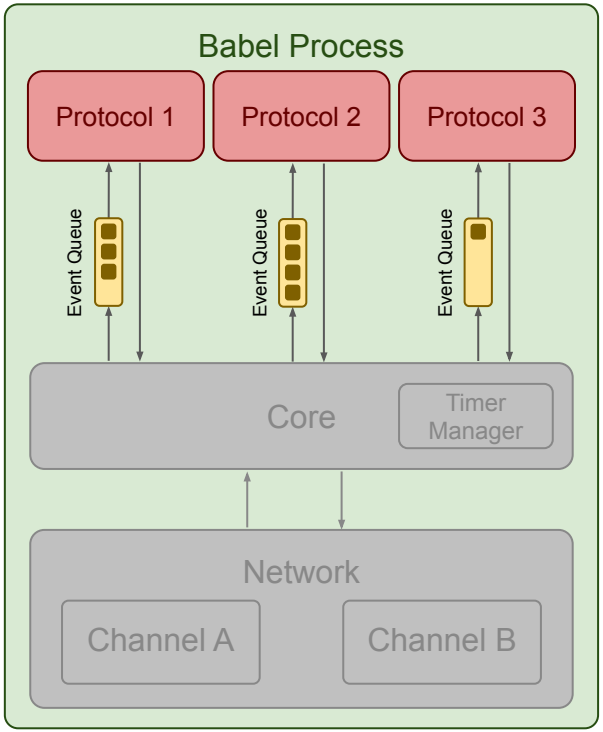
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.
- Received events are added to the event queue.
- **Dedicated thread** per protocol handles events in **serial** fashion.
- Developer defines and implements **callbacks** for each **event.**

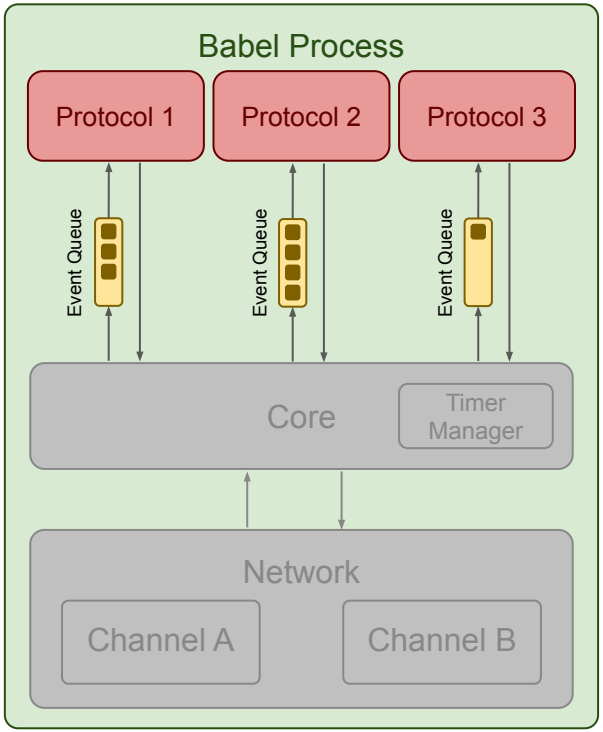
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.
- Received events are added to the event queue.
- **Dedicated thread** per protocol handles events in **serial** fashion.
- Developer defines and implements **callbacks** for each **event.**
- **State-machine** that evolves by processing events.

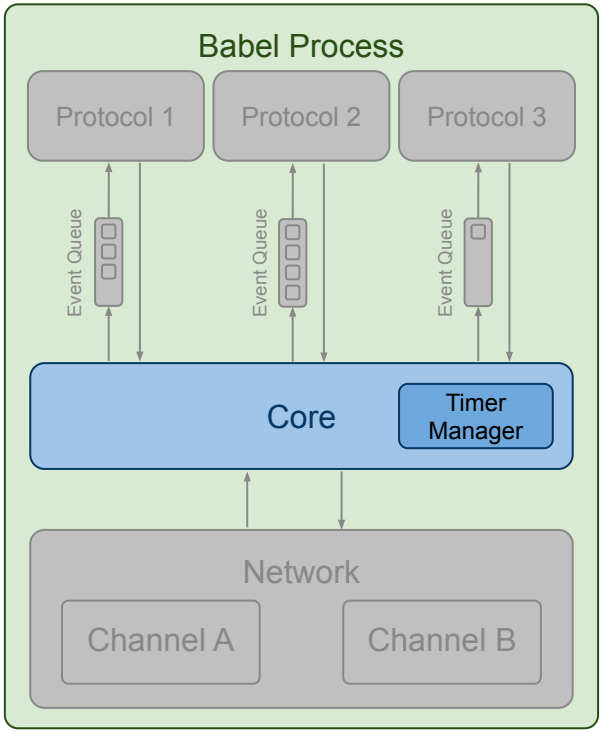
Babel: Architecture



Protocols:

- Encode the behaviour of the distributed algorithm.
- Received events are added to the event queue.
- **Dedicated thread** per protocol handles events in **serial** fashion.
- Developer defines and implements **callbacks** for each **event.**
- **State-machine** that evolves by processing events.
- Communication by **message passing** (no shared state).

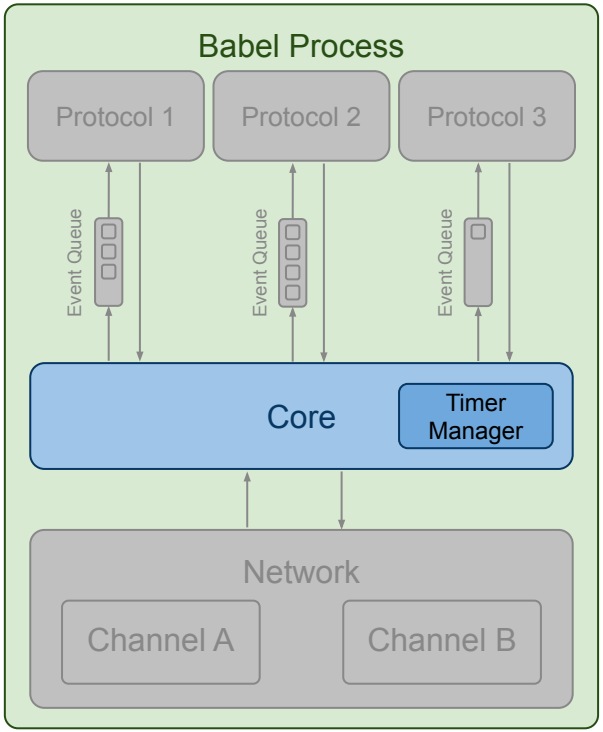
Babel: Architecture



Core:

- Mediates all interactions between protocols.

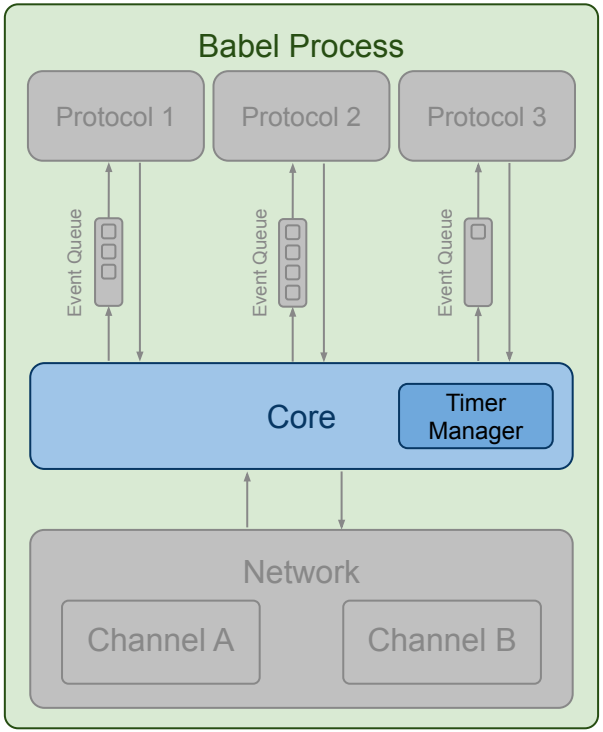
Babel: Architecture



Core:

- Mediates all interactions between protocols.
  - Exchanges events **between protocols** in the same process.

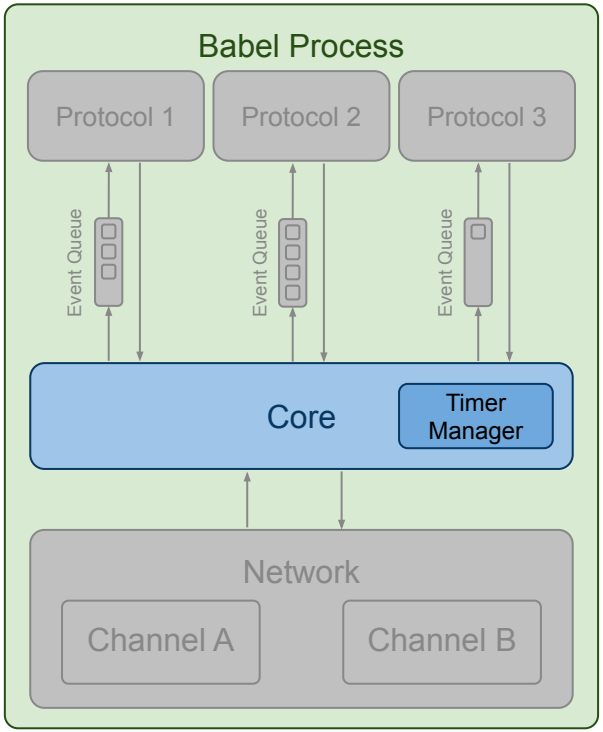
Babel: Architecture



Core:

- Mediates all interactions between protocols.
  - Exchanges events **between protocols** in the same process.
  - Delivers outgoing **network messages** from protocols to the network channel and incoming ones to the correct protocol.

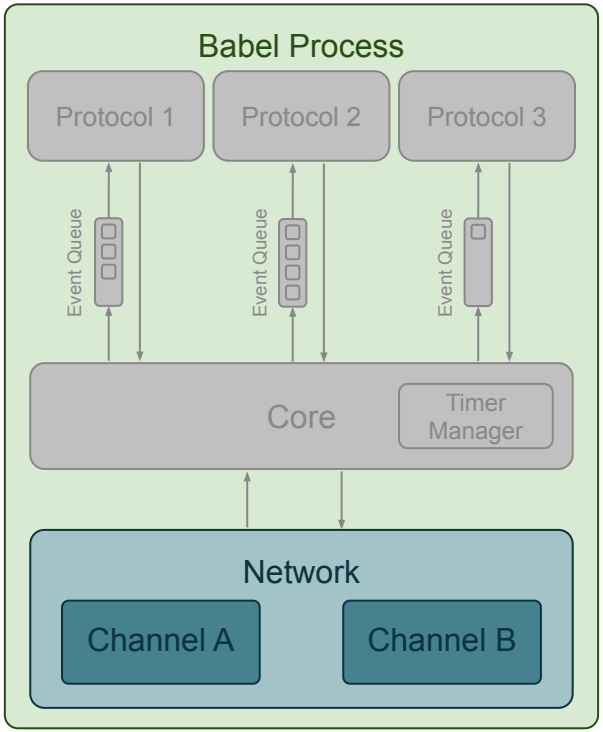
Babel: Architecture



Core:

- Mediates all interactions between protocols.
  - Exchanges events **between protocols** in the same process.
  - Delivers outgoing **network messages** from protocols to the network channel and incoming ones to the correct protocol.
  - Tracks **timers** and delivers timer events to protocols.

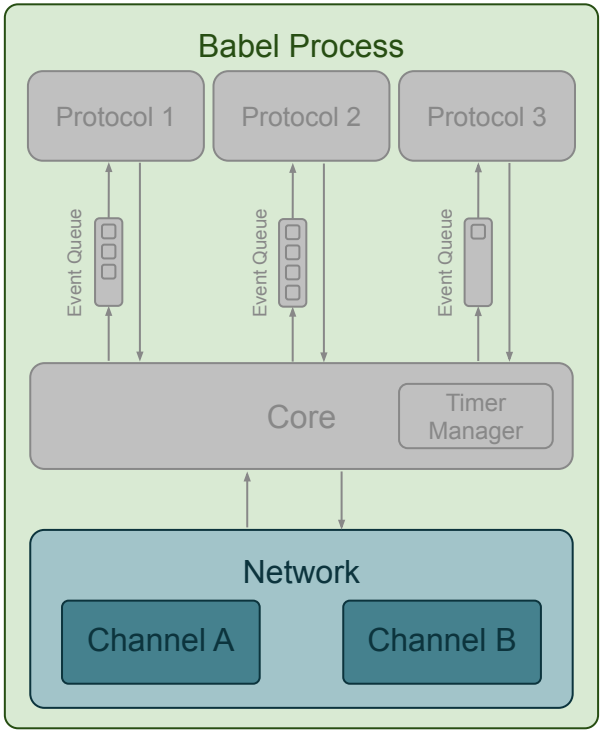
Babel: Architecture



Network channels:

- **Abstract complexity** of dealing with networking.

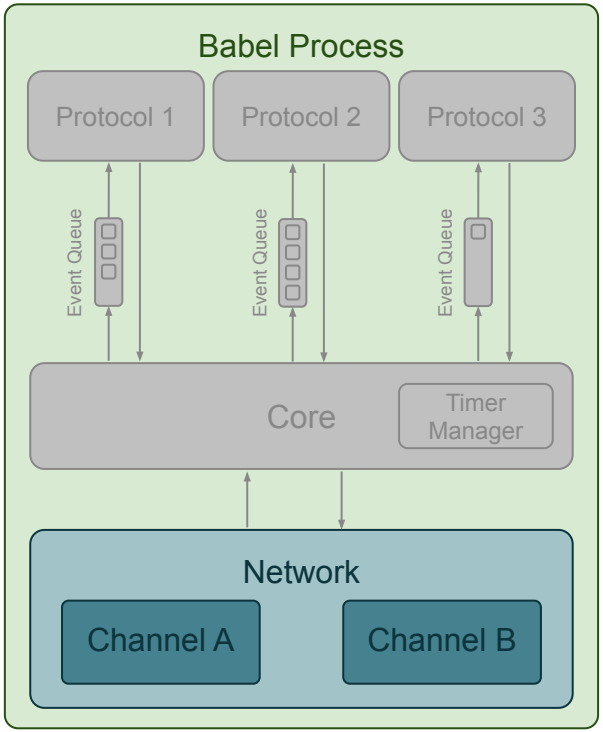
Babel: Architecture



Network channels:

- **Abstract complexity** of dealing with networking.
- Each provides different **guarantees and behaviour.**

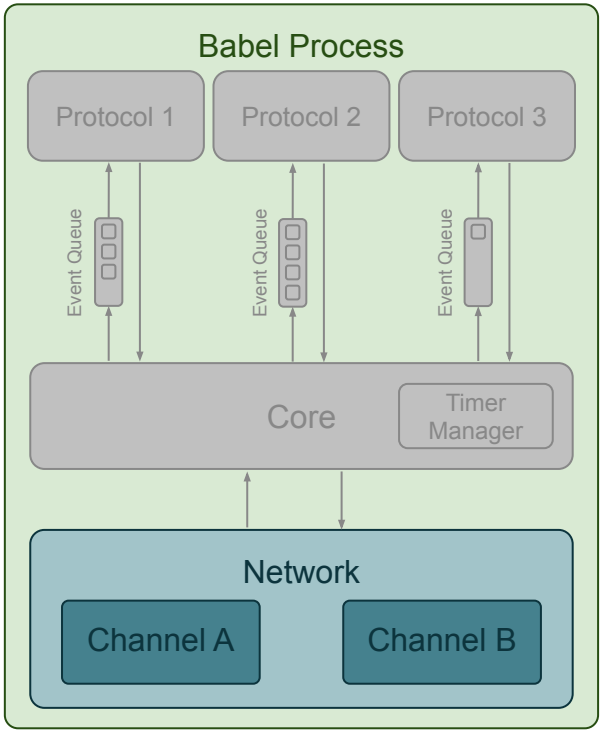
Babel: Architecture



Network channels:

- **Abstract complexity** of dealing with networking.
- Each provides different **guarantees and behaviour.**
  - e.g.: explicit and automatic acknowledgement of messages; transparent creation of multiple TCP connections; failure detectors.

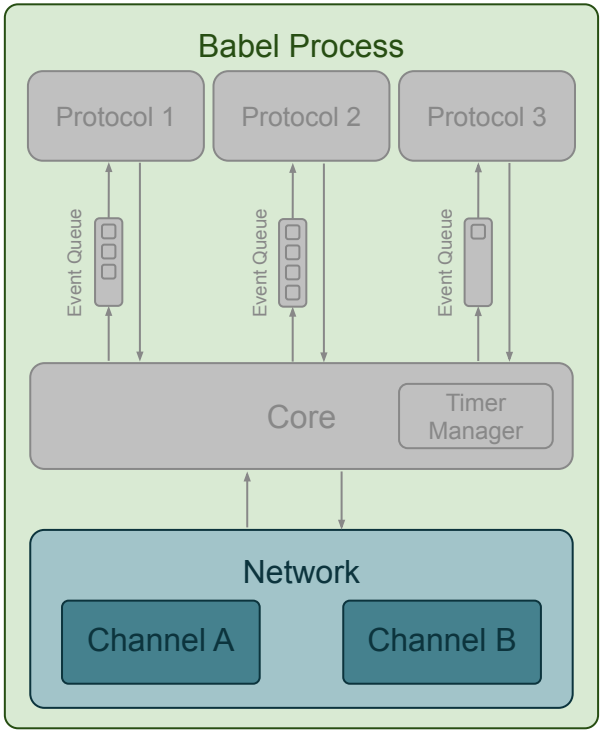
Babel: Architecture



Network channels:

- **Abstract complexity** of dealing with networking.
- Each provides different **guarantees and behaviour.**
  - e.g.: explicit and automatic acknowledgement of messages; transparent creation of multiple TCP connections; failure detectors.
- Developers can **create additional channels** that provides **specific abstractions** for their protocols.

Babel: Architecture



Network channels:

- **Abstract complexity** of dealing with networking.
- Each provides different **guarantees and behaviour.**
  - e.g.: explicit and automatic acknowledgement of messages; transparent creation of multiple TCP connections; failure detectors.
- Developers can **create additional channels** that provides **specific abstractions** for their protocols.
- Protocols can use **multiple channels**, channels can be **shared** by multiple protocols.

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    ...
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Protocols extend an abstract Java class

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    ...
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Protocols extend an abstract Java class

Unique identifier for each protocol

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    ...
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Protocols extend an abstract Java class

Unique identifier for each protocol

Init function initializes the protocol

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    ...
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Protocols extend an abstract Java class

Unique identifier for each protocol

Init function initializes the protocol

Event handlers control protocol flow

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    registerMessageHandler(HelloMsg.ID, this::uponHelloMsg);
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Protocols extend an abstract Java class

Unique identifier for each protocol

Init function initializes the protocol

Event handlers control protocol flow (must be registered)

Babel: API

```
public class HelloProtocol extends GenericProtocol {

  public HelloProtocol() {
    super("Hello", 100);
  }

  public void init (Properties props) {
    registerMessageHandler(HelloMsg.ID, this::uponHelloMsg);
    sendMessage(new HelloMsg(), contactNode);
  }

  private void uponHelloMsg(HelloMsg msg, Host from){
    ...
  }
}
```

Implementing protocols consists (mostly) in creating and registering handlers for events

Protocols extend an abstract Java class

Unique identifier for each protocol

Init function initializes the protocol

Event handlers control protocol flow (must be registered)

Babel: API

- Protocols handle 3 types of Events:
  - Timers
  - Inter-Protocol Communication
  - Networking

Babel: API

- Protocols handle 3 types of Events:
  - **Timers**
    - Periodic/Timed actions are a common part of distributed applications.
  - Inter-Protocol Communication
  - Networking

Babel: API - Timers

```
public class CounterTimer extends ProtoTimer {

  public static final short ID = 101;
  int counter;

  public CounterTimer(int initValue) {
    super(ID);
    counter = initValue;
  }

  public void decCounter() {
    return --counter;
  }
}
```

Extends an abstract Java class

Babel: API - Timers

```
public class CounterTimer extends ProtoTimer {

  public static final short ID = 101;
  int counter;

  public CounterTimer(int initValue) {
    super(ID);
    counter = initValue;
  }

  public void decCounter() {
    return --counter;
  }
}
```

Extends an abstract Java class

Requires unique ID

Babel: API - Timers

```
public class CounterTimer extends ProtoTimer {

  public static final short ID = 101;
  int counter;

  public CounterTimer(int initValue) {
    super(ID);
    counter = initValue;
  }

  public void decCounter() {
    return --counter;
  }
}
```

Extends an abstract Java class

Requires unique ID

Can have any internal logic

Babel: API - Timers

```
public class CountdownProtocol extends GenericProtocol {

  public CountdownProtocol() {
    super("Countdown", 100);
  }

  @Override
  public void init (Properties props) {
    registerTimerHandler(CounterTimer.ID, this::handleCounter);
    setupPeriodicTimer(new CounterTimer(10), 1000, 300);
  }

  private void handleCounter(CounterTimer timer, long tId){
    doImportantStuff();
    if(timer.decCounter() == 0) cancelTimer(tId);
  }
}
```

Babel: API - Timers

```
public class CountdownProtocol extends GenericProtocol {

  public CountdownProtocol() {
    super("Countdown", 100);
  }

  @Override
  public void init (Properties props) {
    registerTimerHandler(CounterTimer.ID, this::handleCounter);
    setupPeriodicTimer(new CounterTimer(10), 1000, 300);
  }

  private void handleCounter(CounterTimer timer, long tId){
    doImportantStuff();
    if(timer.decCounter() == 0) cancelTimer(tId);
  }
}
```

Setup periodic or "one-shot" timers

Babel: API - Timers

```
public class CountdownProtocol extends GenericProtocol {

  public CountdownProtocol() {
    super("Countdown", 100);
  }

  @Override
  public void init (Properties props) {
    registerTimerHandler(CounterTimer.ID, this::handleCounter);
    setupPeriodicTimer(new CounterTimer(10), 1000, 300);
  }

  private void handleCounter(CounterTimer timer, long tId){
    doImportantStuff();
    if(timer.decCounter() == 0) cancelTimer(tId);
  }
}
```

Setup periodic or "one-shot" timers

Setup and register handlers for timers

Babel: API - Timers

```
public class CountdownProtocol extends GenericProtocol {

  public CountdownProtocol() {
    super("Countdown", 100);
  }

  @Override
  public void init (Properties props) {
    registerTimerHandler(CounterTimer.ID, this::handleCounter);
    setupPeriodicTimer(new CounterTimer(10), 1000, 300);
  }

  private void handleCounter(CounterTimer timer, long tId){
    doImportantStuff();
    if(timer.decCounter() == 0) cancelTimer(tId);
  }
}
```

Setup periodic or "one-shot" timers

Setup and register handlers for timers

Babel: API - Timers

```
public class CountdownProtocol extends GenericProtocol {

  public CountdownProtocol() {
    super("Countdown", 100);
  }

  @Override
  public void init (Properties props) {
    registerTimerHandler(CounterTimer.ID, this::handleCounter);
    setupPeriodicTimer(new CounterTimer(10), 1000, 300);
  }

  private void handleCounter(CounterTimer timer, long tId){
    doImportantStuff();
    if(timer.decCounter() == 0) cancelTimer(tId);
  }
}
```

Setup periodic or "one-shot" timers

Setup and register handlers for timers

Cancel timers

Babel: API

- Protocols handle 3 types of Events:
  - Timers
    - Periodic/Timed actions are a common part of distributed applications.
  - Inter-Protocol Communication
  - Networking

Babel: API

- Protocols handle 3 types of Events:
  - Timers
    - Periodic/Timed actions are a common part of distributed applications.
  - **Inter-Protocol Communication**
    - Distributed applications require multiple different protocols to coordinate.
    - One-to-one **requests/replies**
    - One-to-many **notifications**
  - Networking

Babel: API - Inter-protocol communication

```
public class ViewChange extends ProtoNotification {

  public static final short ID = 101;
  Peer peer;

  public ViewChange(Peer p) {
    super(ID);
    peer = p;
  }
}
```

Extends an abstract Java class

Babel: API - Inter-protocol communication

```
public class ViewChange extends ProtoNotification {

  public static final short ID = 101;
  Peer peer;

  public ViewChange(Peer p) {
    super(ID);
    peer = p;
  }
}
```

Extends an abstract Java class

Requires unique ID

Babel: API - Inter-protocol communication

```
public class ViewChange extends ProtoNotification {

  public static final short ID = 101;
  Peer peer;

  public ViewChange(Peer p) {
    super(ID);
    peer = p;
  }
}
```

Extends an abstract Java class

Requires unique ID

Can have any internal logic

Babel: API - Inter-protocol communication

Example:

- **Membership Protocol**
  - Keeps a view with the current membership
- **Dissemination Protocol**
  - Requests current membership and subscribes to changes

Babel: API - Inter-protocol communication

```
public class DisseminationProtocol extends GenericProtocol {

  public DisseminationProtocol() { super("Disseminator", 200); }

  public void init (Properties props) {
    subscribeNotification(ViewChange.ID, this::onViewChange);
    registerReplyHandler(ViewReply.ID, this::onViewReply);
    sendRequest(new ViewRequest(), MembershipProtocol.ID);
  }

  private void onViewReply(ViewReply reply, short from) {...}
  private void onViewChange(ViewChange not, short emitter){...}
}
```

Babel: API - Inter-protocol communication

```
public class DisseminationProtocol extends GenericProtocol {

  public DisseminationProtocol() { super("Disseminator", 200); }

  public void init (Properties props) {
    subscribeNotification(ViewChange.ID, this::onViewChange);
    registerReplyHandler(ViewReply.ID, this::onViewReply);
    sendRequest(new ViewRequest(), MembershipProtocol.ID);
  }

  private void onViewReply(ViewReply reply, short from) {...}
  private void onViewChange(ViewChange not, short emitter){...}
}
```

Setup and register handlers

Babel: API - Inter-protocol communication

```
public class DisseminationProtocol extends GenericProtocol {

  public DisseminationProtocol() { super("Disseminator", 200); }

  public void init (Properties props) {
    subscribeNotification(ViewChange.ID, this::onViewChange);
    registerReplyHandler(ViewReply.ID, this::onViewReply);
    sendRequest(new ViewRequest(), MembershipProtocol.ID);
  }

  private void onViewReply(ViewReply reply, short from) {...}
  private void onViewChange(ViewChange not, short emitter){...}
}
```

Setup and register handlers

Sending requests/replies

Babel: API - Inter-protocol communication

```
public class MembershipProtocol extends GenericProtocol {

  public static final short ID = 300

  public MembershipProtocol() { super("Membership", ID); }

  public void init (Properties props) {
    registerRequestHandler(ViewRequest.ID, this::onViewRequest);
  }

  private void onViewRequest(ViewRequest request, short from) {
    sendReply(new ViewReply(currentView), from);
  }

  private void onConnectionEstablished(Peer peer){
    triggerNotification(new ViewChange(peer));
  }
}
```

Babel: API - Inter-protocol communication

```
public class MembershipProtocol extends GenericProtocol {

  public static final short ID = 300

  public MembershipProtocol() { super("Membership", ID); }

  public void init (Properties props) {
    registerRequestHandler(ViewRequest.ID, this::onViewRequest);
  }

  private void onViewRequest(ViewRequest request, short from) {
    sendReply(new ViewReply(currentView), from);
  }

  private void onConnectionEstablished(Peer peer){
    triggerNotification(new ViewChange(peer));
  }
}
```

Replies to received requests

Babel: API - Inter-protocol communication

```
public class MembershipProtocol extends GenericProtocol {

  public static final short ID = 300

  public MembershipProtocol() { super("Membership", ID); }

  public void init (Properties props) {
    registerRequestHandler(ViewRequest.ID, this::onViewRequest);
  }

  private void onViewRequest(ViewRequest request, short from) {
    sendReply(new ViewReply(currentView), from);
  }

  private void onConnectionEstablished(Peer peer){
    triggerNotification(new ViewChange(peer));
  }
}
```

Replies to received requests

Notifies all subscribers

Babel: API - Inter-protocol communication

```
public class MembershipProtocol extends GenericProtocol {

  public static final short ID = 300

  public MembershipProtocol() { super("Membership", ID); }

  public void init (Properties props) {
    registerRequestHandler(ViewRequest.ID, this::onViewRequest);
  }

  private void onViewRequest(ViewRequest request, short from) {
    sendReply(new ViewReply(currentView), from);
  }

  private void onConnectionEstablished(Peer peer){
    triggerNotification(new ViewChange(peer));
  }
}
```

Replies to received requests

Notifies all subscribers

No destination protocol required

Babel: API

- Protocols handle 3 types of Events:
  - Timers
    - Periodic/Timed actions are a common part of distributed applications.
  - Inter-Protocol Communication
    - Distributed applications require multiple different protocols to coordinate.
    - One-to-one requests/replies
    - One-to-many notifications
  - Networking

Babel: API

- Protocols handle 3 types of Events:
  - Timers
    - Periodic/Timed actions are a common part of distributed applications.
  - Inter-Protocol Communication
    - Distributed applications require multiple different protocols to coordinate.
    - One-to-one requests/replies
    - One-to-many notifications
  - **Networking**
    - Distributed application need networking to be distributed.

Babel: API - Networking

```
public class PingMsg extends ProtoMessage {
  public final static short MSG_ID = 401;
  private final long timestamp;

  public PingMsg(long timestamp) {
    super(MSG_ID);
    this.timestamp = timestamp;
  }

  public static ISerializer<PingMsg> serializer = new ISerializer<>() {
    public void serialize(PingMsg msg, ByteBuf out) {
      out.writeLong(msg.timestamp);
    }

    public PingMsg deserialize(ByteBuf in){
      return new PingMsg(in.readLong());
    }
  }
}
```

Network messages extend generic class

Babel: API - Networking

```
public class PingMsg extends ProtoMessage {
  public final static short MSG_ID = 401;
  private final long timestamp;

  public PingMsg(long timestamp) {
    super(MSG_ID);
    this.timestamp = timestamp;
  }

  public static ISerializer<PingMsg> serializer = new ISerializer<>() {
    public void serialize(PingMsg msg, ByteBuf out) {
      out.writeLong(msg.timestamp);
    }

    public PingMsg deserialize(ByteBuf in){
      return new PingMsg(in.readLong());
    }
  }
}
```

Network messages extend generic class

Can have any internal logic

Babel: API - Networking

```
public class PingMsg extends ProtoMessage {
  public final static short MSG_ID = 401;
  private final long timestamp;

  public PingMsg(long timestamp) {
    super(MSG_ID);
    this.timestamp = timestamp;
  }

  public static ISerializer<PingMsg> serializer = new ISerializer<>() {
    public void serialize(PingMsg msg, ByteBuf out) {
      out.writeLong(msg.timestamp);
    }

    public PingMsg deserialize(ByteBuf in){
      return new PingMsg(in.readLong());
    }
  }
}
```

Network messages extend generic class

Can have any internal logic

Require a serializer to encode and decode

Babel: API - Networking

```
public class PingMsg extends ProtoMessage {
  public final static short MSG_ID = 401;
  private final long timestamp;

  public PingMsg(long timestamp) {
    super(MSG_ID);
    this.timestamp = timestamp;
  }

  public static ISerializer<PingMsg> serializer = new ISerializer<>() {
    public void serialize(PingMsg msg, ByteBuf out) {
      out.writeLong(msg.timestamp);
    }

    public PingMsg deserialize(ByteBuf in){
      return new PingMsg(in.readLong());
    }
  }
}
```

Network messages extend generic class

Can have any internal logic

Require a serializer to encode and decode

Babel: API - Networking

```
public class PingMsg extends ProtoMessage {
  public final static short MSG_ID = 401;
  private final long timestamp;

  public PingMsg(long timestamp) {
    super(MSG_ID);
    this.timestamp = timestamp;
  }

  public static ISerializer<PingMsg> serializer = new ISerializer<>() {
    public void serialize(PingMsg msg, ByteBuf out) {
      out.writeLong(msg.timestamp);
    }

    public PingMsg deserialize(ByteBuf in){
      return new PingMsg(in.readLong());
    }
  }
}
```

Network messages extend generic class

Can have any internal logic

Require a serializer to encode and decode

Babel: API - Networking

Example:

- **Ping Pong Protocol**
  - Chooses a random peer to send a Ping message
  - Peer responds with Pong message
  - Repeat

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Babel: API - Networking

Create channel

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Create channel

Register serializer for each message

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Create channel

Register serializer for each message

Register handler for each message

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Create channel

Register serializer for each message

Register handler for each message

Register other channel events

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {
  List<Host> peers;

  public PingPongProto() { super("PingPong", 400); }

  public void init (Properties props) {
    int cId = createChannel(TCPChannel.Name, props);
    registerMessageSerializer(cId, PingMsg.ID, PingMsg.serializer);
    registerMessageHandler(cId, PingMsg.ID, this::uponPing);
    registerChannelEventHandler(cId, OutConnUp.EVENT_ID,
        this::uponOutConnUp);
    openConnection(random(peers));
  }
}
```

Create channel

Register serializer for each message

Register handler for each message

Register other channel events

Create connections

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {

  private void uponOutConnUp(OutConnectionUp ev, ...){
    sendMessage(new PingMsg(currTime), ev.getPeer());
  }

  private void uponPing(PingMsg msg, Host from, ...){
    sendMessage(new PongMsg(msg.getValue()), from);
  }

  private void uponPong(PongMsg msg, Host from, ...){
    closeConnection(from);
    openConnection(random(peers));
  }

}
```

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {

  private void uponOutConnUp(OutConnectionUp ev, ...){
    sendMessage(new PingMsg(currTime), ev.getPeer());
  }

  private void uponPing(PingMsg msg, Host from, ...){
    sendMessage(new PongMsg(msg.getValue()), from);
  }

  private void uponPong(PongMsg msg, Host from, ...){
    closeConnection(from);
    openConnection(random(peers));
  }

}
```

Handle channel events

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {

  private void uponOutConnUp(OutConnectionUp ev, ...){
    sendMessage(new PingMsg(currTime), ev.getPeer());
  }

  private void uponPing(PingMsg msg, Host from, ...){
    sendMessage(new PongMsg(msg.getValue()), from);
  }

  private void uponPong(PongMsg msg, Host from, ...){
    closeConnection(from);
    openConnection(random(peers));
  }

}
```

Handle channel events

Send messages

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {

  private void uponOutConnUp(OutConnectionUp ev, ...){
    sendMessage(new PingMsg(currTime), ev.getPeer());
  }

  private void uponPing(PingMsg msg, Host from, ...){
    sendMessage(new PongMsg(msg.getValue()), from);
  }

  private void uponPong(PongMsg msg, Host from, ...){
    closeConnection(from);
    openConnection(random(peers));
  }

}
```

Handle channel events

Send messages

Handle received messages

Babel: API - Networking

```
public class PingPongProto extends GenericProtocol {

  private void uponOutConnUp(OutConnectionUp ev, ...){
    sendMessage(new PingMsg(currTime), ev.getPeer());
  }

  private void uponPing(PingMsg msg, Host from, ...){
    sendMessage(new PongMsg(msg.getValue()), from);
  }

  private void uponPong(PongMsg msg, Host from, ...){
    closeConnection(from);
    openConnection(random(peers));
  }

}
```

Handle channel events

Send messages

Handle received messages

Open/close connections

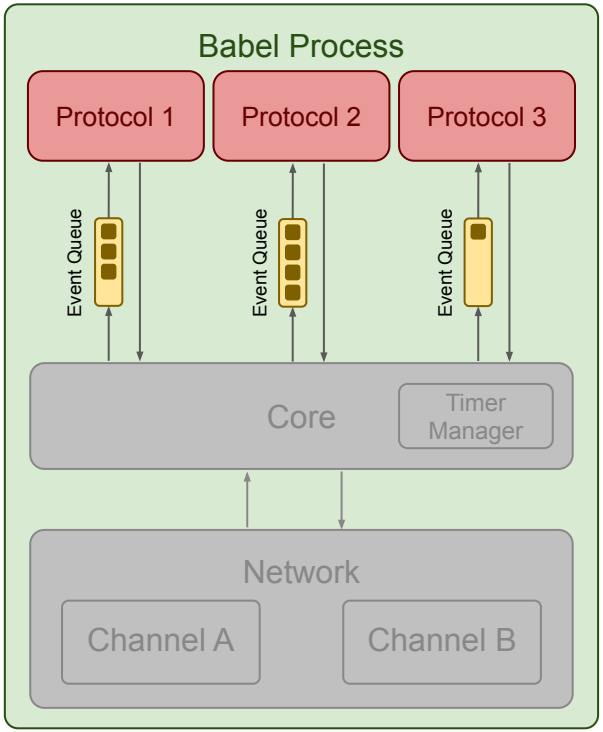
Babel: Summary

- **Developers** only implement **handlers for events**, and register them.
- Implementation **closely matches** typical algorithm specification (**pseudo-code**).
- **Shields developers** from the complexities of concurrency, multiplexing and networking
- Abstractions are sufficiently **generic** to support a wide variety of algorithms.
- Capable of supporting **production-ready performant** implementations.

Babel: Summary

- **Developers** only implement **handlers for events**, and register them.
- Implementation **closely matches** typical algorithm specification (**pseudo-code**).
- **Shields developers** from the complexities of concurrency, multiplexing and networking
- Abstractions are sufficiently **generic** to support a wide variety of algorithms.
- Capable of supporting **production-ready performant** implementations.

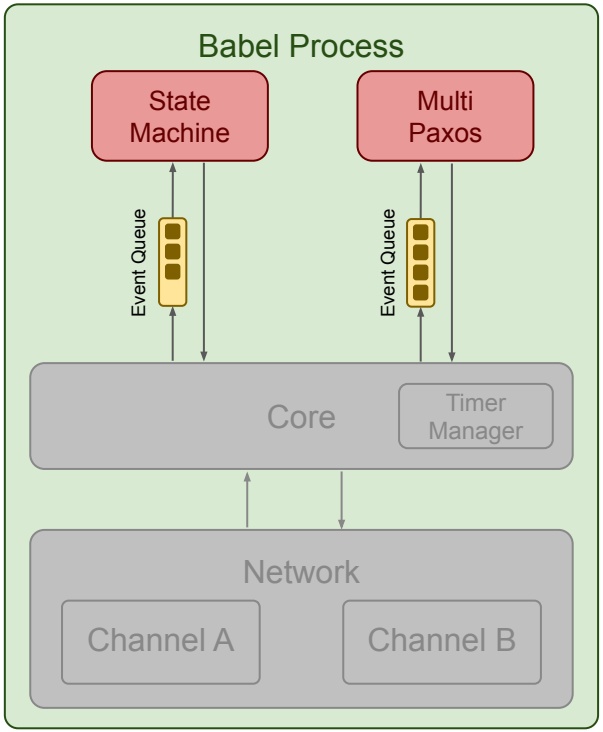
Class Example - MultiPaxos



Two Protocols

- State Machine
- MultiPaxos

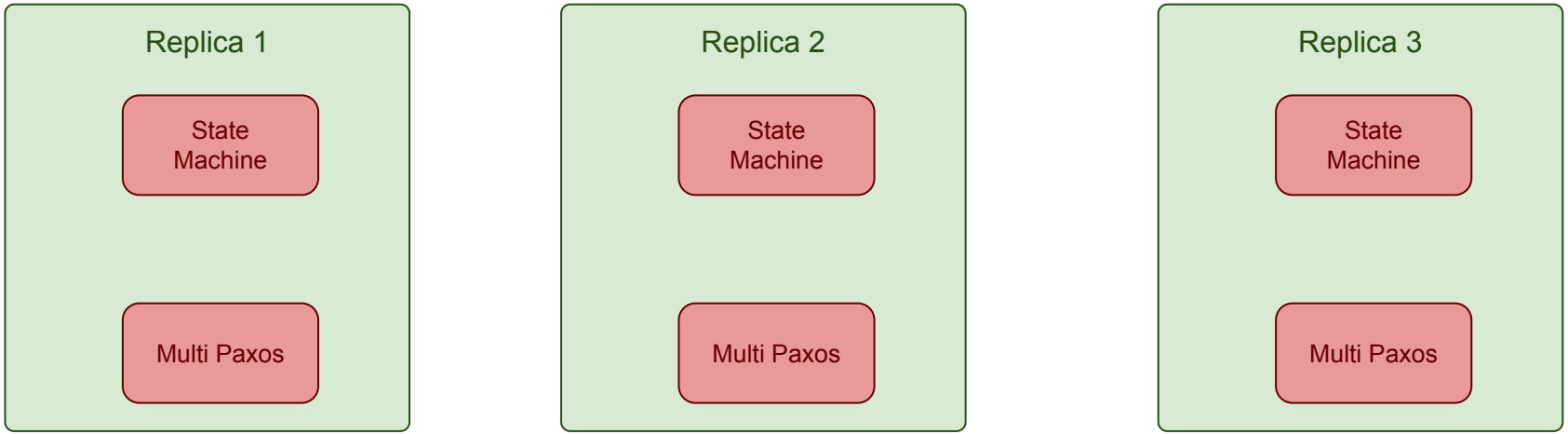
Class Example - MultiPaxos



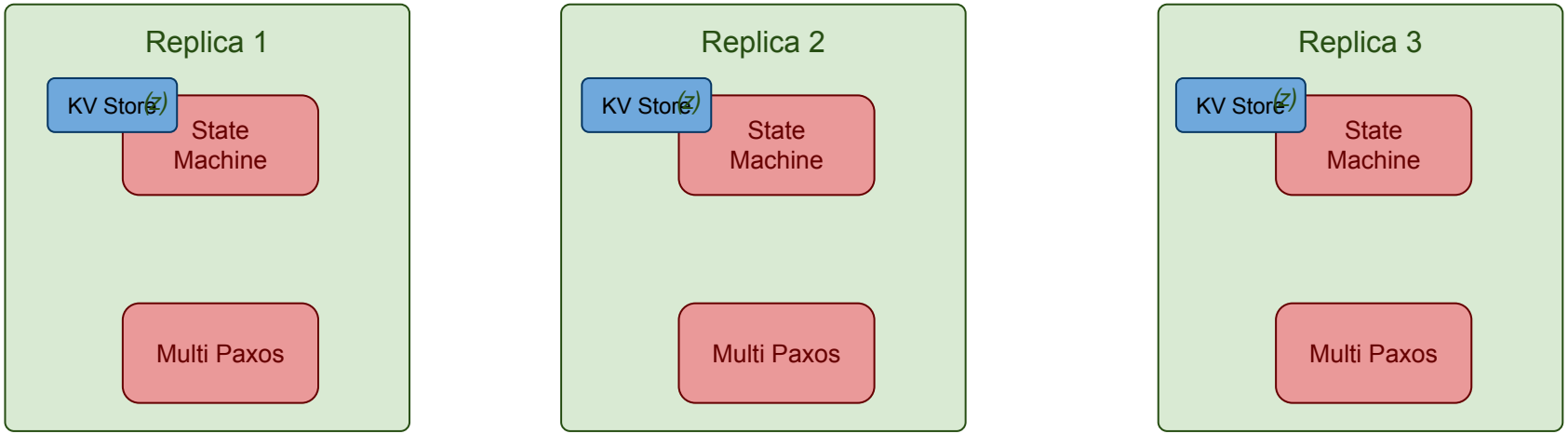
Two Protocols

- State Machine
- MultiPaxos

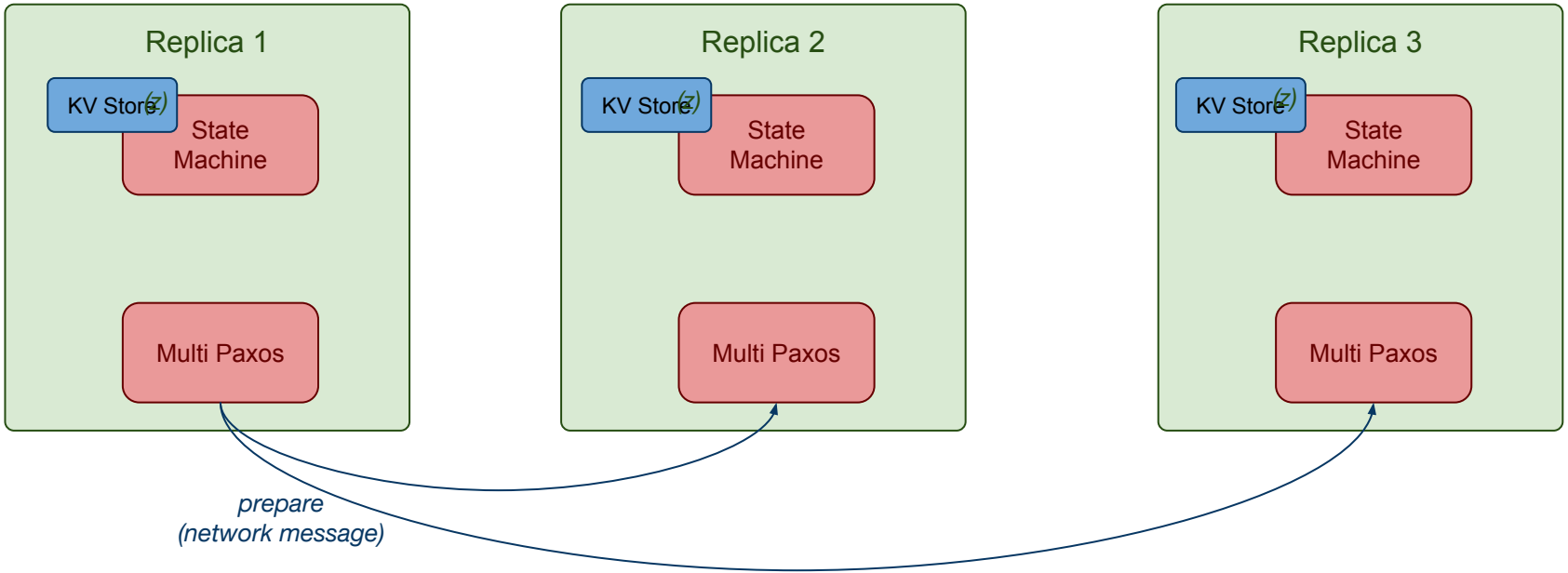
Class Example - MultiPaxos



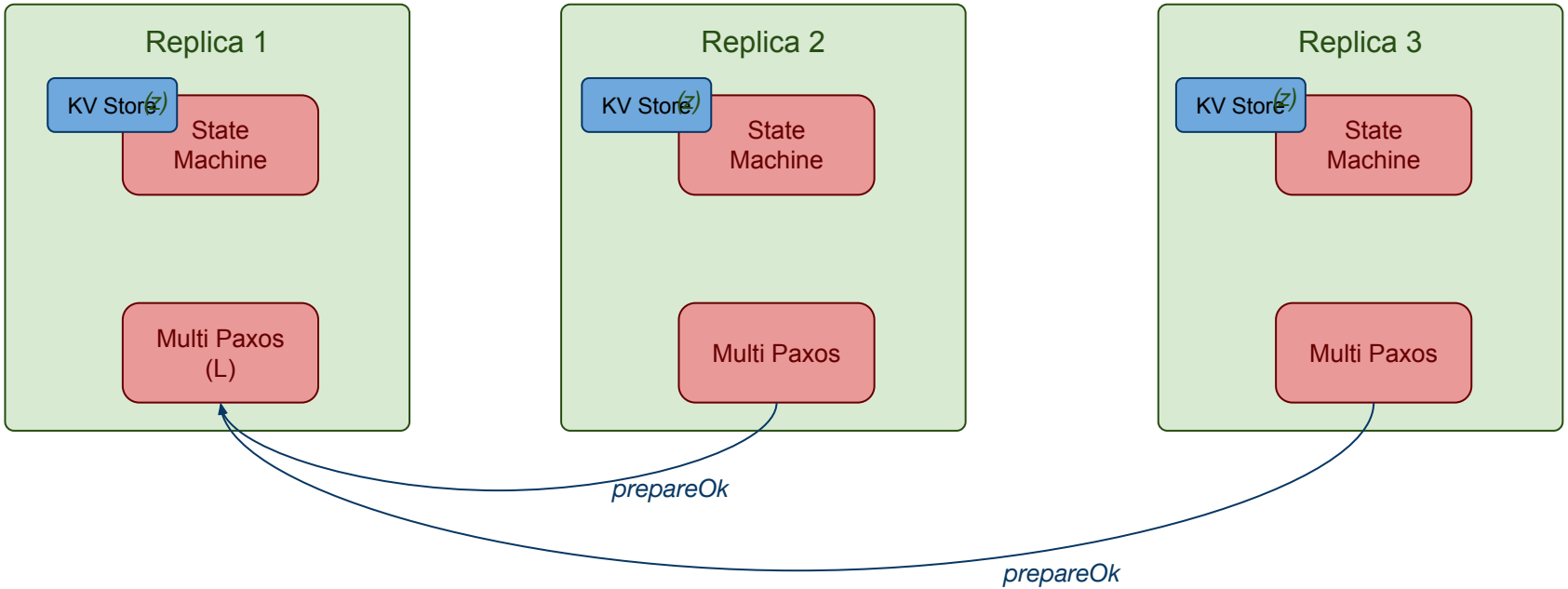
Class Example - MultiPaxos



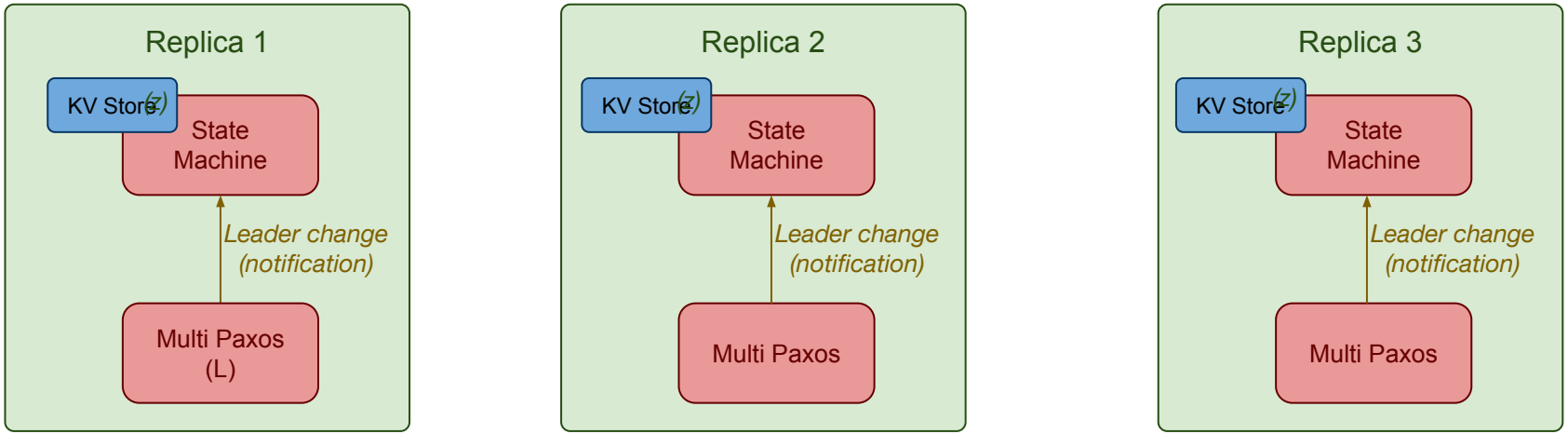
Class Example - MultiPaxos



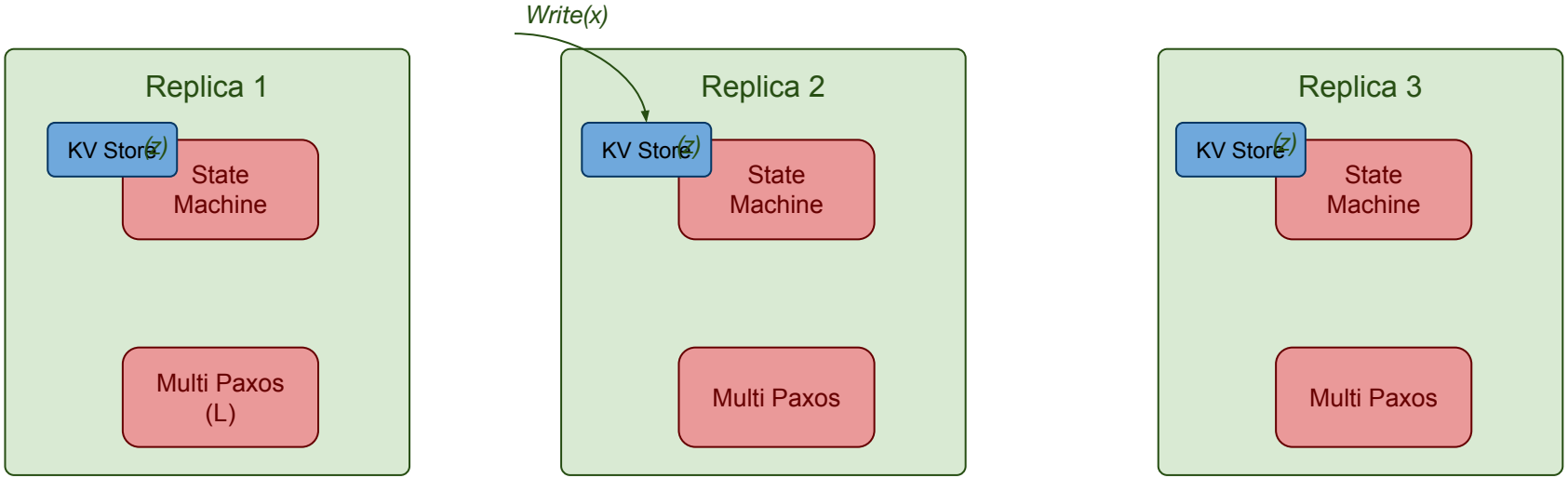
Class Example - MultiPaxos



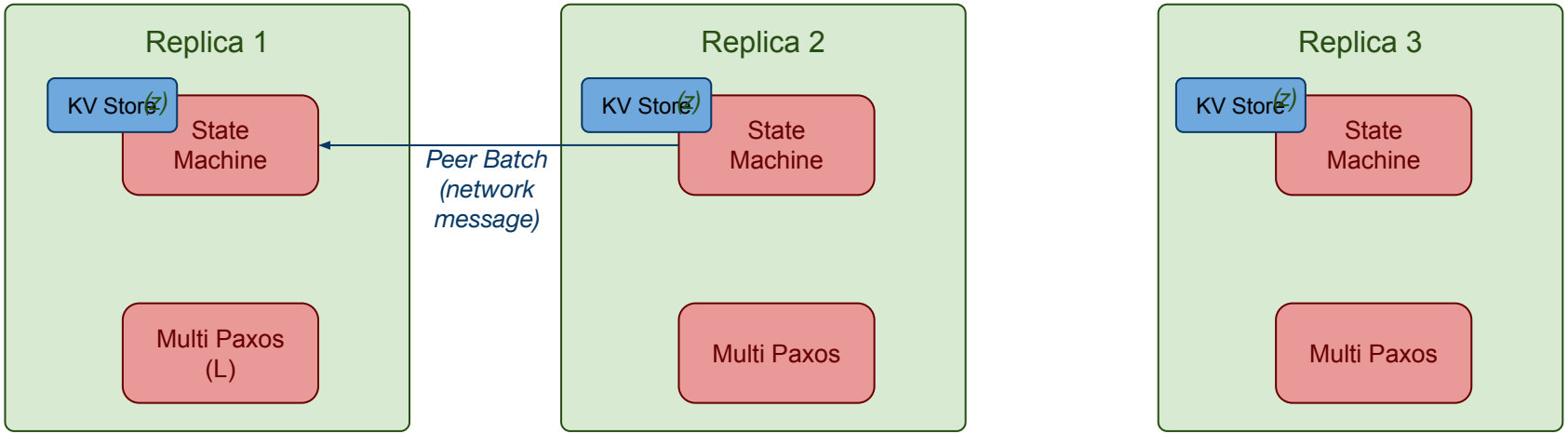
Class Example - MultiPaxos



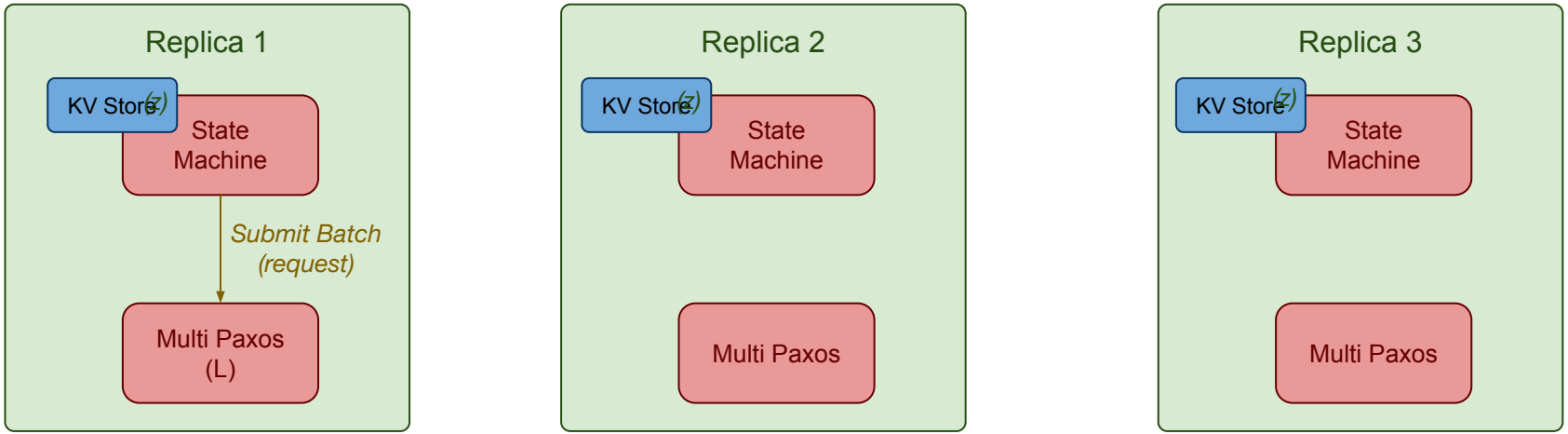
Class Example - MultiPaxos



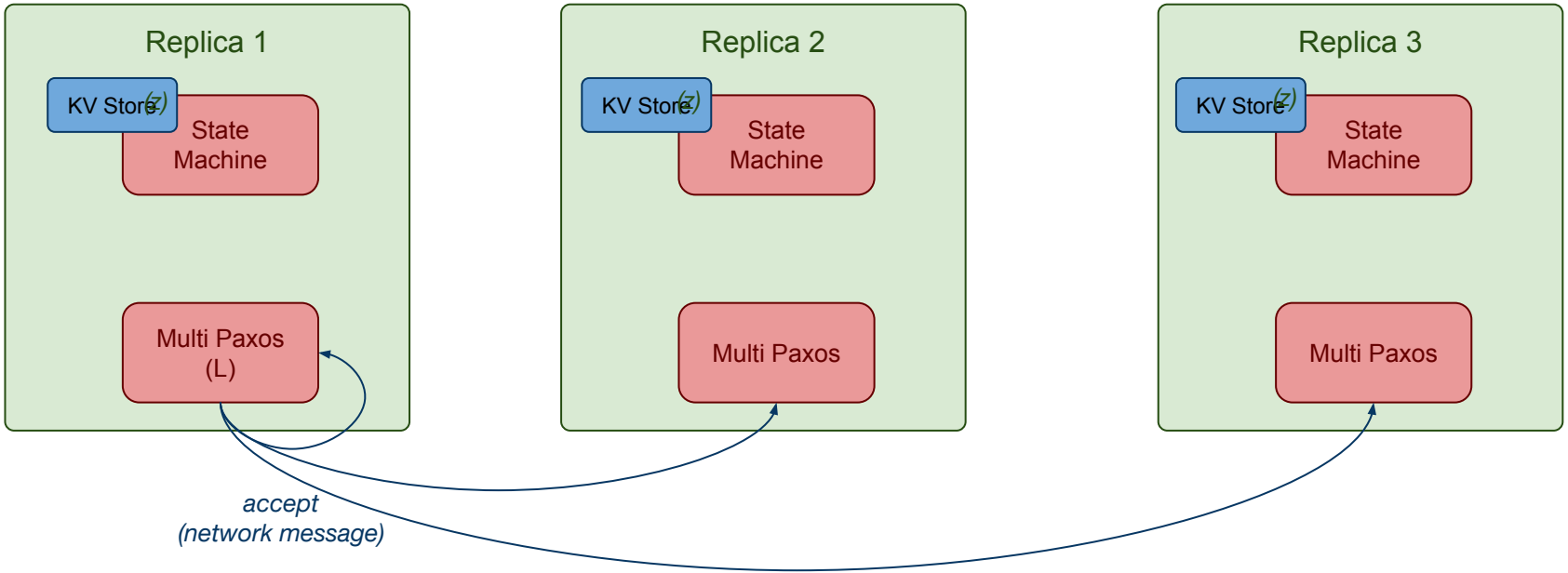
Class Example - MultiPaxos



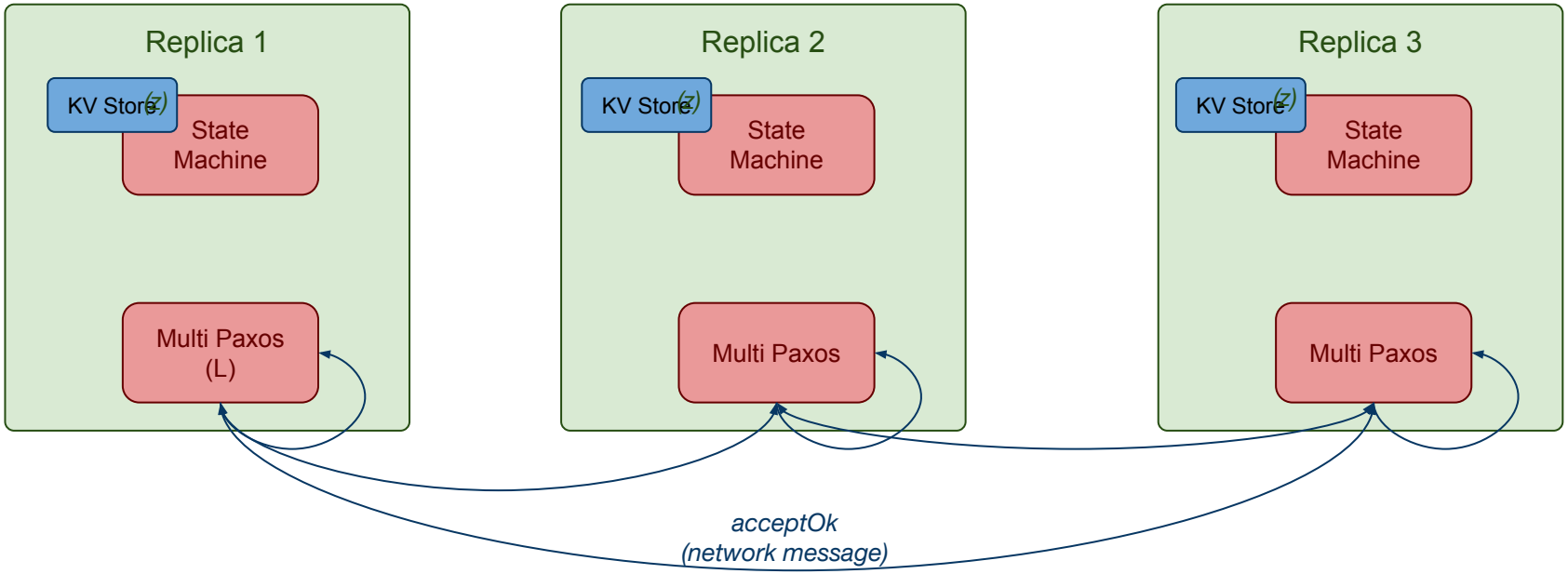
Class Example - MultiPaxos



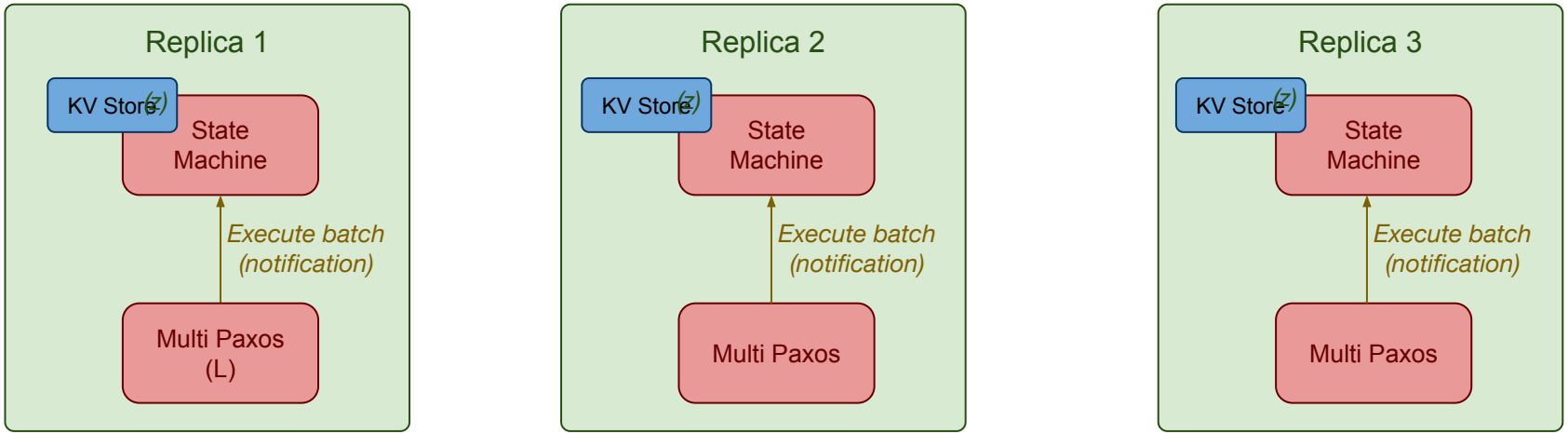
Class Example - MultiPaxos



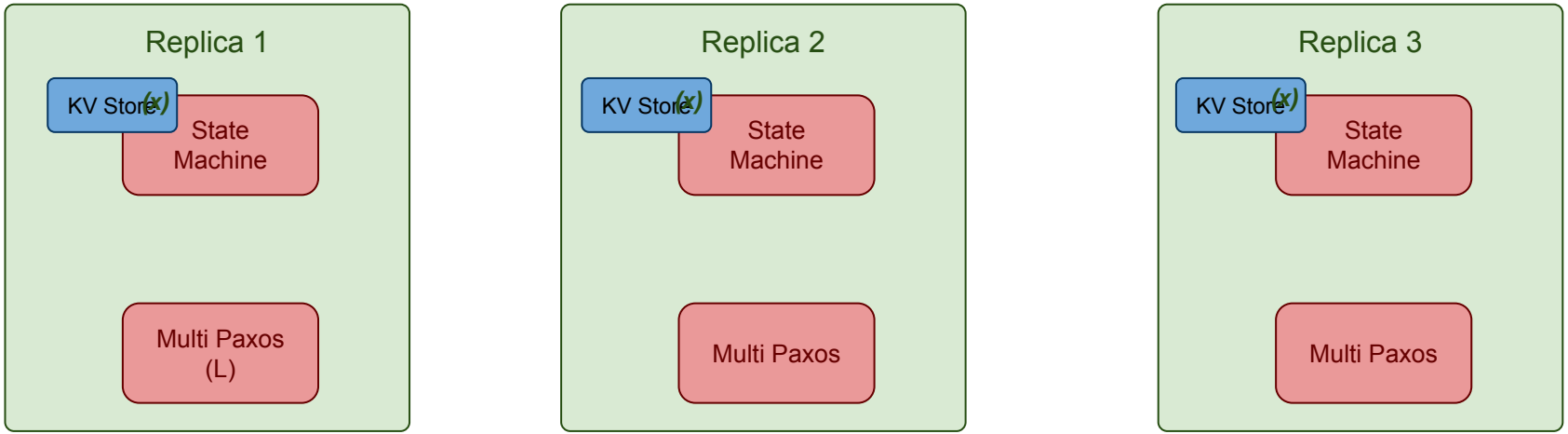
Class Example - MultiPaxos



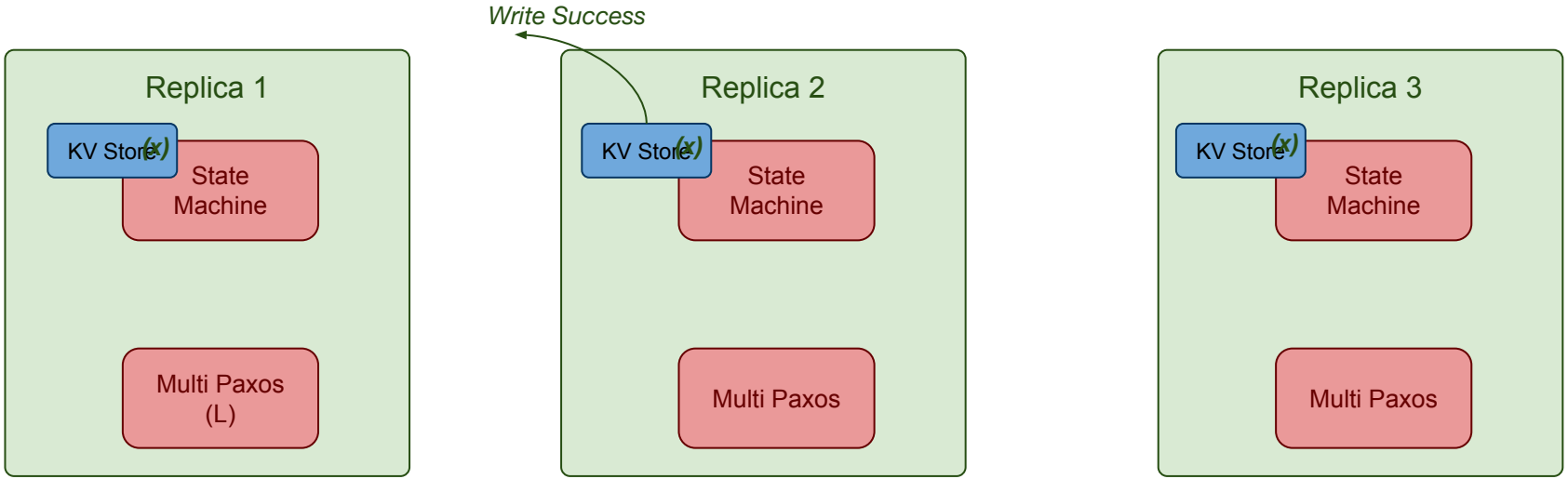
Class Example - MultiPaxos



Class Example - MultiPaxos



Class Example - MultiPaxos



Babel: A framework for developing performant and dependable distributed protocols

Pedro Fouto, Pedro Ákos Costa,
Nuno Preguiça, João Leitão

SRDS 2022

https://github.com/pfouto/babel-core
https://github.com/pedroAkos/babel-case-studies

FCT
Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR
NOVA
NOVA SCHOOL OF SCIENCE & TECHNOLOGY