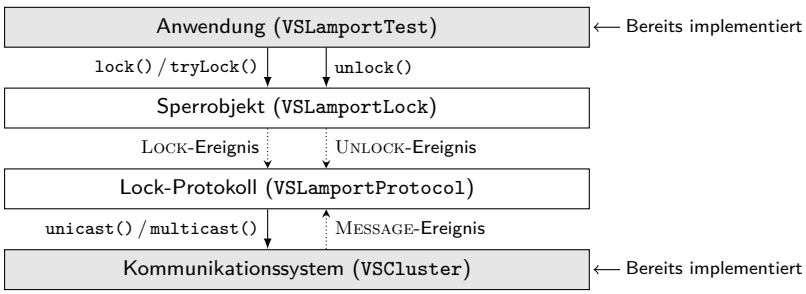
# Übungsaufgabe #6: Verteilte Synchronisation

In dieser Übungsaufgabe soll die Koordinierung von Zugriffen auf eine gemeinsame Ressource in einem verteilten System betrachtet werden. Konkretes Ziel ist es dabei, das Lock-Protokoll von Lamport zu implementieren, das die Synchronisation mehrerer Prozesse ohne den Einsatz eines zentralen Koordinators ermöglicht.




Die Umsetzung des Sperrmechanismus soll neben der bereits implementierten Anwendung `VSLamportTest` und dem ebenfalls vorhandenen Kommunikationssystem `VSCluster` bei jedem Teilnehmer zwei weitere Hauptbestandteile umfassen: Ein Sperrobjekt `VSLamportLock`, das die Schnittstelle zur Anwendung realisiert, sowie eine Protokollkomponente `VSLamportProtocol`, die für die Abstimmung mit ihren entsprechenden Gegenstücken auf anderen Rechnern zuständig ist und festlegt, wer den kritischen Abschnitt als nächstes betreten darf. Die Interaktion zwischen Sperrobjekt und Protokollkomponente findet ausschließlich über Ereignisse statt, die vom Protokoll sequentiell bearbeitet werden, um konsistente Entscheidungen garantieren zu können. Außer auf die durch das Sperrobjekt ausgelösten Ereignisse für die Lock-Anforderung (LOCK) und -Freigabe (UNLOCK), reagiert die Protokollkomponente darüber hinaus auch auf MESSAGE-Ereignisse, mit deren Hilfe das Kommunikationssystem das Eintreffen von Nachrichten signalisiert. Die Vorgaben für diese Übungsaufgabe, inklusive bereits existierender Komponenten und Testfälle, sind im Pub-Verzeichnis unter `/proj/i4vs/pub/aufgabe6` hinterlegt.

## 6.1 Sperrobjekt (für alle)

Zentrale Aufgabe des Sperrobjekts ist es, die in Form von Methodenaufrufen eintreffenden Anfragen der lokalen Instanz einer verteilten Anwendung entgegen zu nehmen und sie in entsprechende Ereignisse für die Protokollkomponente zu übersetzen [Folie 6.3 : 3]. Hierzu verfügt die Klasse `VSLamportLock` über folgende Anwendungsschnittstelle:

```
public class VSLamportLock {
    public void lock();
    public void unlock();
}
```

Mit der Methode `lock()` signalisiert die Anwendung, dass sie den kritischen Abschnitt betreten möchte. Ein Aufruf dieser Methode blockiert den aktuellen Thread, bis ihm das Lock zugewiesen wurde, darf also nicht unterbrechbar sein. Beim Austritt aus dem kritischen Abschnitt gibt die Anwendung die Sperre mittels `unlock()` wieder frei.

Aufgabe:

→ Implementierung des Sperrobjekts in der Klasse `VSLamportLock`

## 6.2 Lock-Protokoll von Lamport (für alle)

Die Klasse `VSLamportProtocol` implementiert den in der Tafelübung präsentierten Lock-Algorithmus von Lamport [Folien 6.2 : 3–5] und stellt die Protokolllogik zur sequentiellen Bearbeitung der auftretenden Ereignisse bereit. Der Algorithmus sieht vor, dass jeder teilnehmende Prozess eine logische Uhr verwaltet und zur Generierung von Zeitstempeln für ausgehende Nachrichten einsetzt. Diese logische Uhr muss gemäß den im Begleitmaterial präzisierten Vorschriften für Lamport-Uhren bei jeder Sende- und Empfangsaktion von Nachrichten aktualisiert werden [Folie 6.1 : 6].

Um den Eintritt in den kritischen Abschnitt zu beantragen, sendet ein Prozess per Multicast eine *Sperranfrage* an alle Teilnehmer im System. Trifft eine solche Nachricht bei einem Empfänger ein, speichert sich dieser die Sperranfrage lokal ab und antwortet unmittelbar mit einer *Bestätigung* an den Absender der Anfrage. Die Entscheidung, wer als nächstes in den kritischen Abschnitt eintreten darf, trifft jeder Prozess für sich selbst auf Basis der ihm vorliegenden Informationen: Die Sperranfrage eines Prozesses wird gewährt, sobald sie (1) den niedrigsten Zeitstempel aller gespeicherten Sperranfragen aufweist und (2) sichergestellt ist, dass auch zukünftig keine andere Sperranfrage mit niedrigerem Zeitstempel mehr eintreffen wird. Letztere Bedingung kann ein Prozess überprüfen, indem er über die höchsten ihm bekannten Lamport-Uhr-Zeitstempel aller Teilnehmer im System Buch führt. Die Bedingung ist erfüllt,

falls alle diese Zeitstempel mindestens so hoch sind wie der Zeitstempel der betroffenen Sperranfrage. Verlässt ein Prozess nach Gewährung einer Sperranfrage zu einem späteren Zeitpunkt den kritischen Abschnitt wieder, signalisiert er dies durch Versand einer *Freigabe* an alle Prozesse. Der Empfang einer Freigabe-Nachricht führt dazu, dass die korrespondierende Sperranfrage aus dem lokalen Speicher eines Prozesses entfernt wird.

```
public class VSLamportProtocol {
    public void init();
    public void event(VSLamportEvent event);
}
```

Die Implementierung der Protokollkomponente soll mindestens zwei Methoden bereitstellen: `init()` wird beim Systemstart von den bereitgestellten Testfällen einmalig aufgerufen und signalisiert den Start der Ereignisbehandlung. Mittels `event()` wird die Protokollkomponente über neue Ereignisse informiert; Ereignisse umfassen neben einem Typ (z. B. MESSAGE) auch ein zugeordnetes Objekt (z. B. die empfangene Nachricht) [Folie 6.3 : 5].

Aufgaben:

→ Implementierung der Protokolllogik in der Klasse `VSLamportProtocol`
→ Testen der Implementierung mit Hilfe der zur Verfügung gestellten Testfälle `simple` und `fancy` [Folie 6.3 : 7]

Hinweise:

- Es ist ausreichend, wenn die Protokollimplementierung lediglich ein einzelnes Sperrobjekt unterstützt.
- Zur Interaktion mit entfernten Prozessen stellt das bereits implementierte `VSCluster`-Kommunikationssystem Methoden für FIFO-Unicast (`unicast()`) bzw. -Multicast (`multicast()`) zur Verfügung [Folie 6.3 : 6].
- Bei identischen Zählerwerten entscheidet die Prozess-ID darüber, welche Sperranfrage Vorrang hat [Folie 6.1 : 8].
- Bestätigungen dienen im Lock-Protokoll von Lamport ausschließlich dazu, anderen Prozessen aktuelle Zeitstempel zukommen zu lassen. Sie stellen keine explizite Eintrittserlaubnis für den kritischen Abschnitt dar.
- Zur Fehlersuche kann der Testfall `debug` hilfreich sein, der auf häufige Implementierungsprobleme testet.

## 6.3 Zeitbeschränkte Sperrversuche (optional für 5,0 ECTS)

Die bisherige Sperrobjektimplementierung aus Teilaufgabe 6.1 zwingt eine Anwendung nach einer Sperranfrage dazu so lange zu blockieren, bis sie tatsächlich in den kritischen Abschnitt eintreten darf. In Fällen, in denen viele Anfragen anderer Prozesse Vorrang haben bzw. der kritische Abschnitt verhältnismäßig lange belegt ist, kann diese Einschränkung zu hohen Blockierungszeiten führen und damit einer Anwendung die Chance nehmen, in der Zwischenzeit eventuell andere, nichtkritische Aufgaben bearbeiten zu können. Ziel dieser Teilaufgabe ist es, der Anwendung eine Möglichkeit zur Verfügung zu stellen, eine obere Schranke für die Blockierungsdauer bei einer Sperranfrage festzulegen. Hierzu soll die Klasse `VSLamportLock` um folgende Methode erweitert werden:

```
    public boolean tryLock(long timeout, TimeUnit unit) throws InterruptedException;
```

Mit Hilfe der Methode `tryLock()` beantragt die Anwendung den Eintritt in den kritischen Abschnitt. Sollte ihr dieser innerhalb des durch die Parameter `timeout` und `unit` spezifierten Zeitintervals gewährt werden, lautet der Rückgabewert `true`. Ist dies nicht der Fall, deblockiert die Methode nach Ablauf der Zeitbeschränkung und gibt `false` zurück. Wird der die Methode aufrufende Thread während der Wartephase unterbrochen, hört `tryLock()` ebenfalls auf, auf die Eintrittserlaubnis zu warten, und wirft stattdessen eine `InterruptedException`.

Aufgaben:

→ Erweiterung der Sperrobjektimplementierung um die Methode `tryLock()`
→ Testen der Implementierung mit Hilfe der zur Verfügung gestellten Testfälle `simple-try` und `fancy-try`

Hinweise:

- Die Implementierung der in dieser Teilaufgabe beschriebenen Funktionalität ist im Normalfall nicht auf `VSLamportLock` beschränkt, sondern erfordert üblicherweise auch Änderungen an `VSLamportProtocol`.
- Den Ereignistypen LOCK, UNLOCK und MESSAGE dürfen, sofern erforderlich, weitere hinzugefügt werden.

## Abgabe: am Mi., 23.07.2025 in der Rechnerübung

Die für diese Übungsaufgabe erstellten Klassen sind in einem Subpackage `vsue.lamport` zusammenzufassen.