Lösungsvorschläge zur Theoretischen Übungsserie B

Verteilte Systeme im HS 2010

Benedikt Ostermaier
(ostermaier@inf.ethz.ch)

Research Group for Distributed Systems
ETH Zürich

19. November 2010
Überarbeitet am 24.11.2010

B1: Lamport-Zeit

Wiederholung: Kausalrelation $\prec$ auf Ereignissen

Es sei x $\prec$ y genau dann, wenn:

1. x und y auf dem gleichen Prozess stattfinden und x vor y kommt, *oder*
2. x ist ein Sendeereignis und y ist das korrespondierende Empfangsereignis *oder*
3. $\exists z$ mit $x \prec z \wedge z \prec y$

B1: Lamport-Zeit (II)

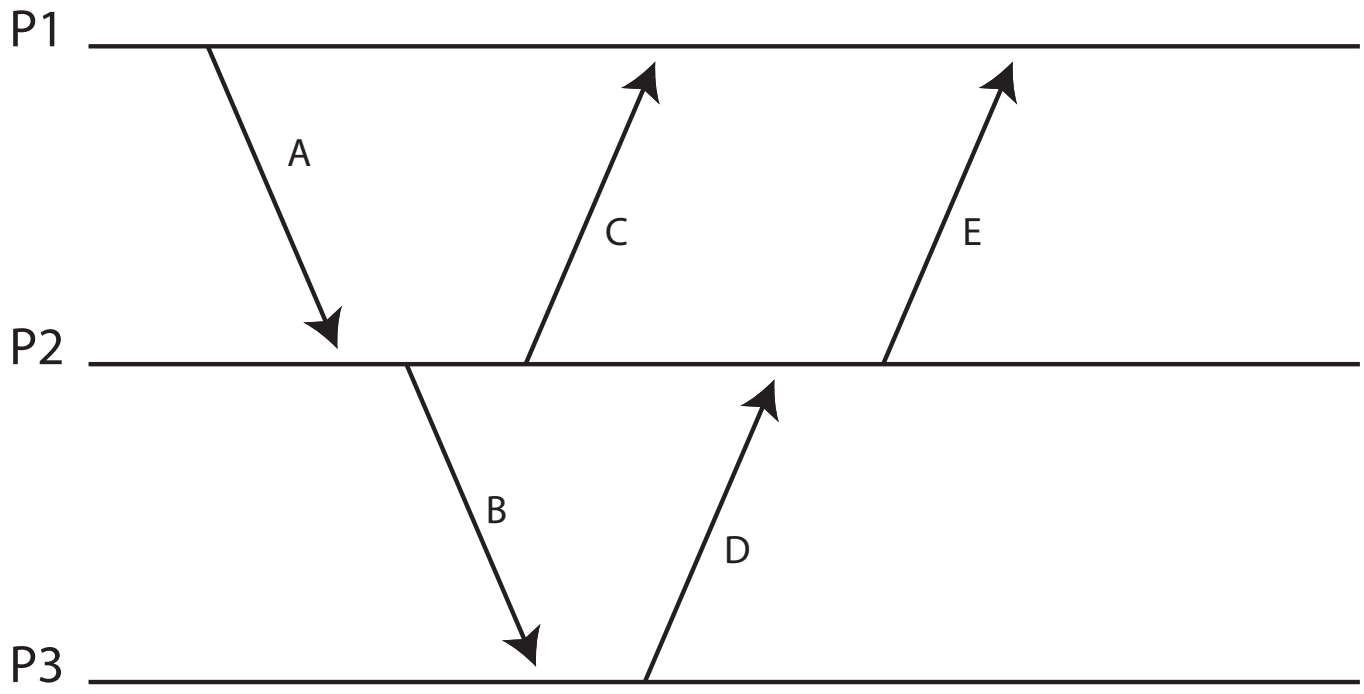
Wiederholung: Uhrenbedingung

- $C : E \to N$ ist Abbildung von Ereignissen auf Zeitstempel
- Es soll gelten: $e \prec e' \Rightarrow C(e) < C(e')$

Protokoll, das Uhrenbedingung implementiert:

- Lokale Uhr (= „Zähler") tickt bei jedem Ereignis
- Sendeereignis: Uhrwert mitsenden (Zeitstempel)
- Empfangsereignis: max(lokale Uhr, Zeitstempel), anschliessend ticken!

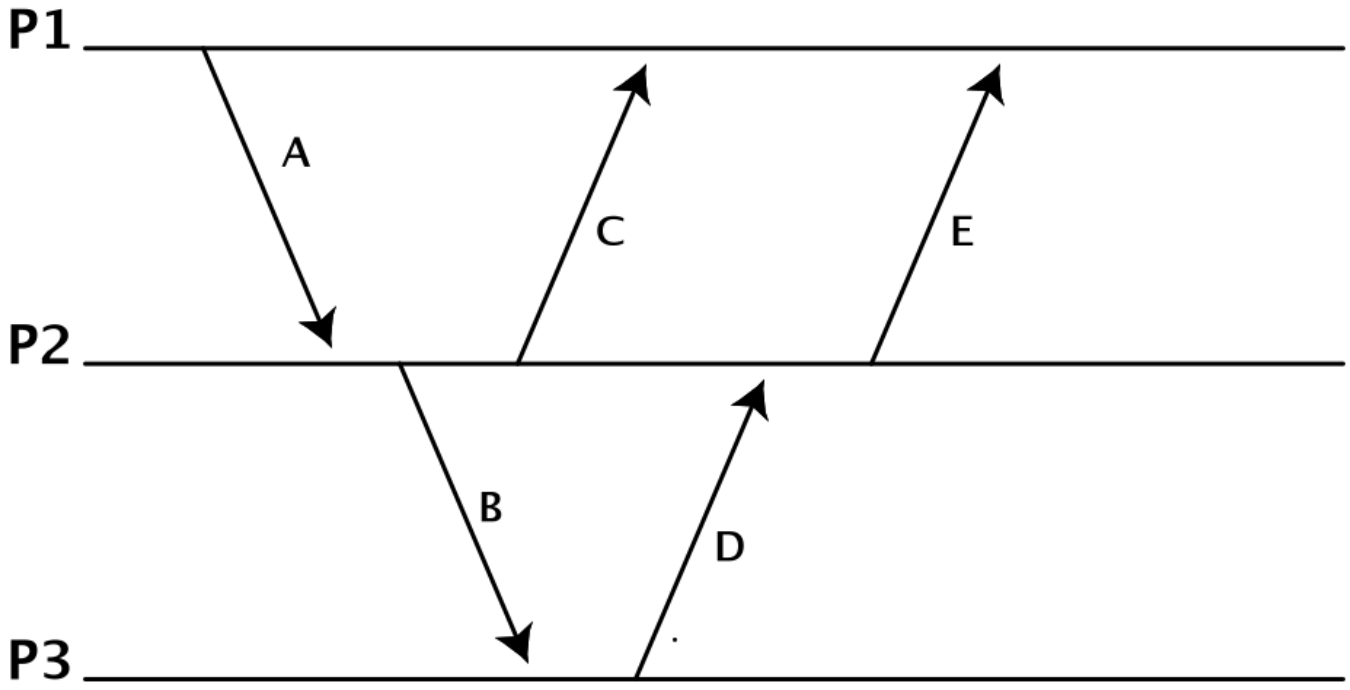
B1: Lamport-Zeit (III)



B.1.1.) Gesucht: Paar von Ereignissen, welche nicht kausal abhängig sind.

Beispiele: (B.receive, C.receive), (C.receive, E.send)

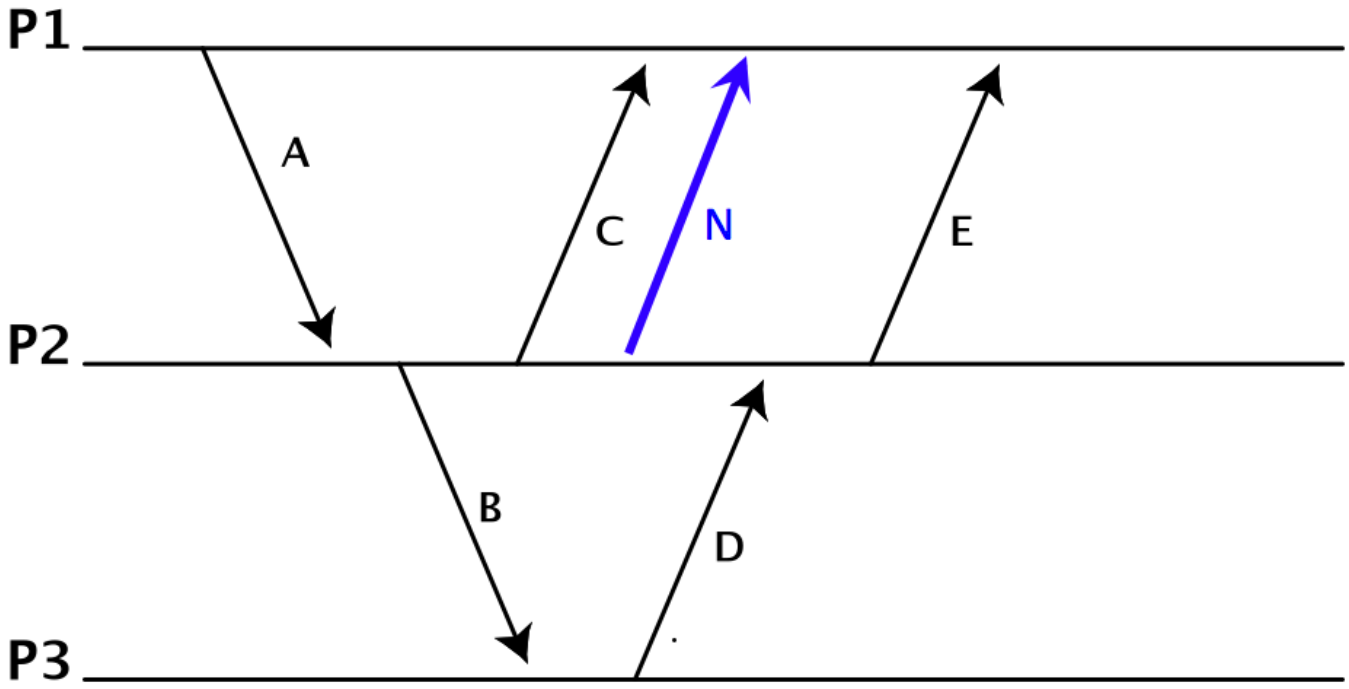
B1: Lamport-Zeit (IV)



B.1.2) Einfügen einer Nachricht N, so dass

- A.receive $\prec$ N.send $\wedge$ C(N.receive) $<$ C(E.send)
- Absender und Empfänger der Nachricht können frei gewählt werden, müssen aber verschieden sein.

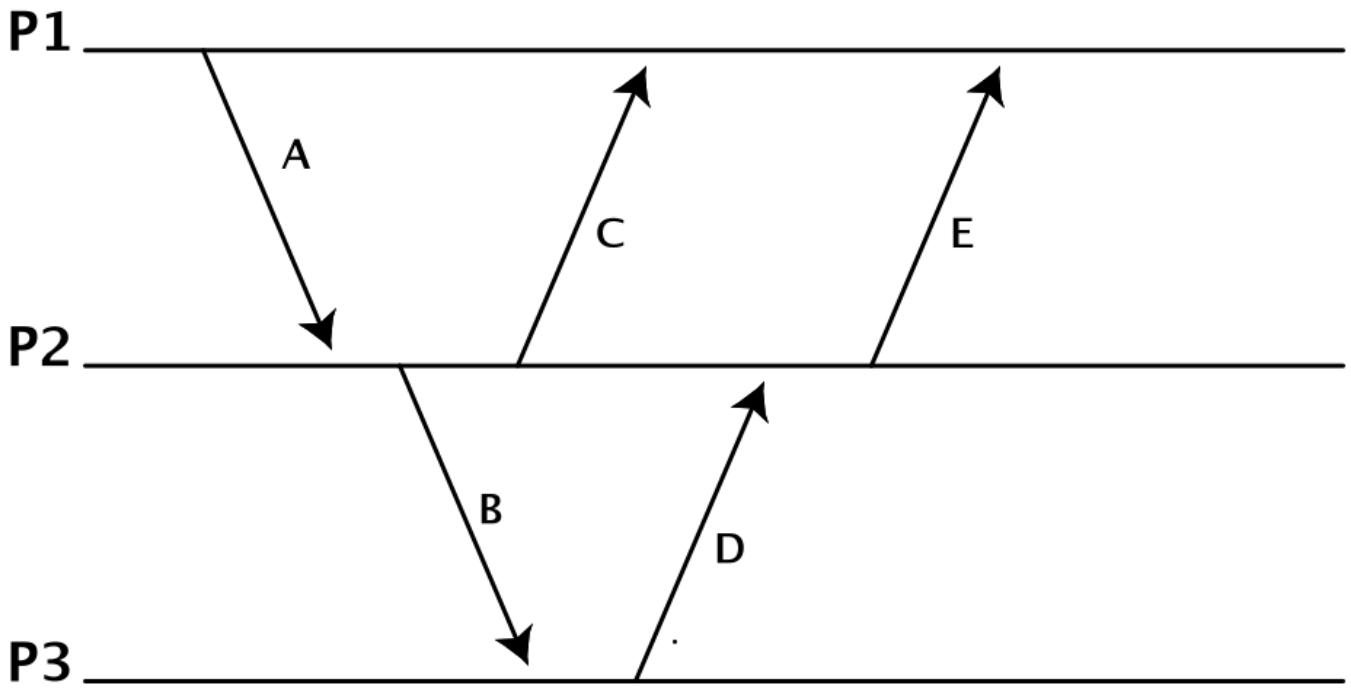
B1: Lamport-Zeit (IV)



B.1.2) Einfügen einer Nachricht N, so dass

- A.receive $\prec$ N.send $\wedge\ C(\text{N.receive}) < C(\text{E.send})$
- Absender und Empfänger der Nachricht können frei gewählt werden, müssen aber verschieden sein.

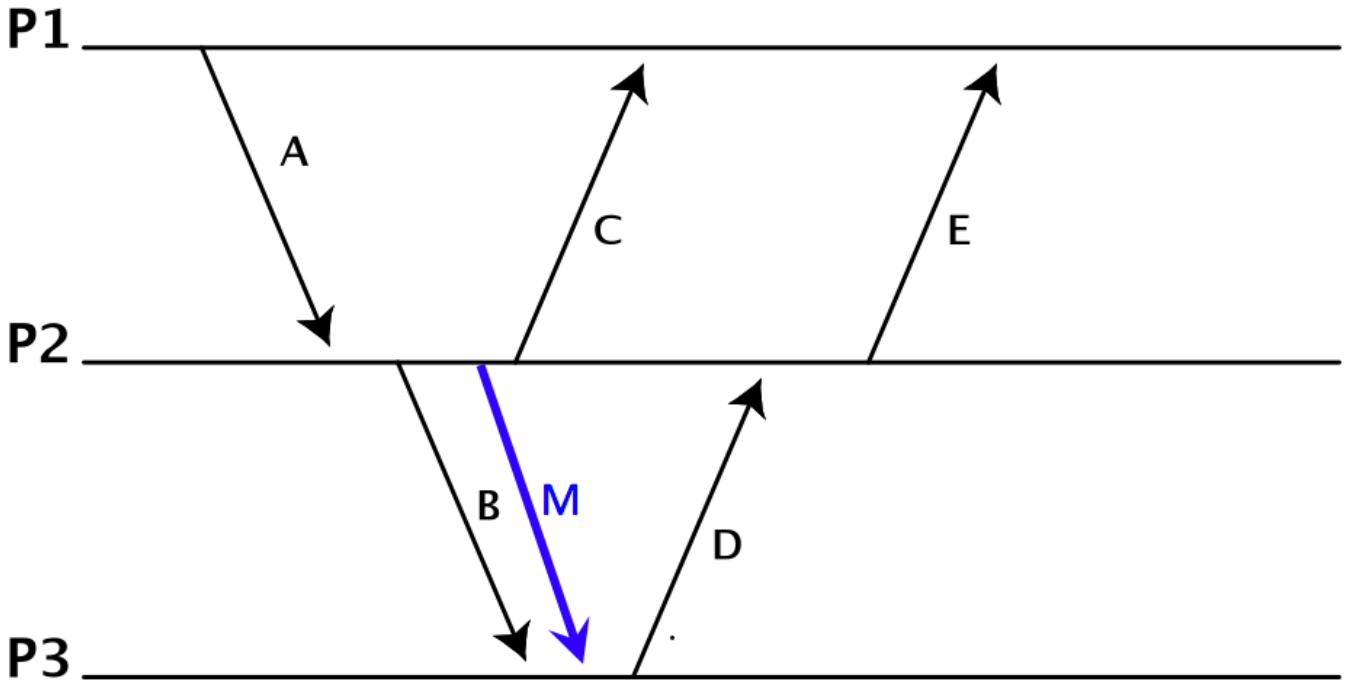
B1: Lamport-Zeit (V)



B.1.3) Einfügen einer Nachricht M, so dass

- M.send $\prec$ C.send $\wedge$ C(B.receive) $<$ C(M.receive)
- Sender: P2, Empfänger: P3

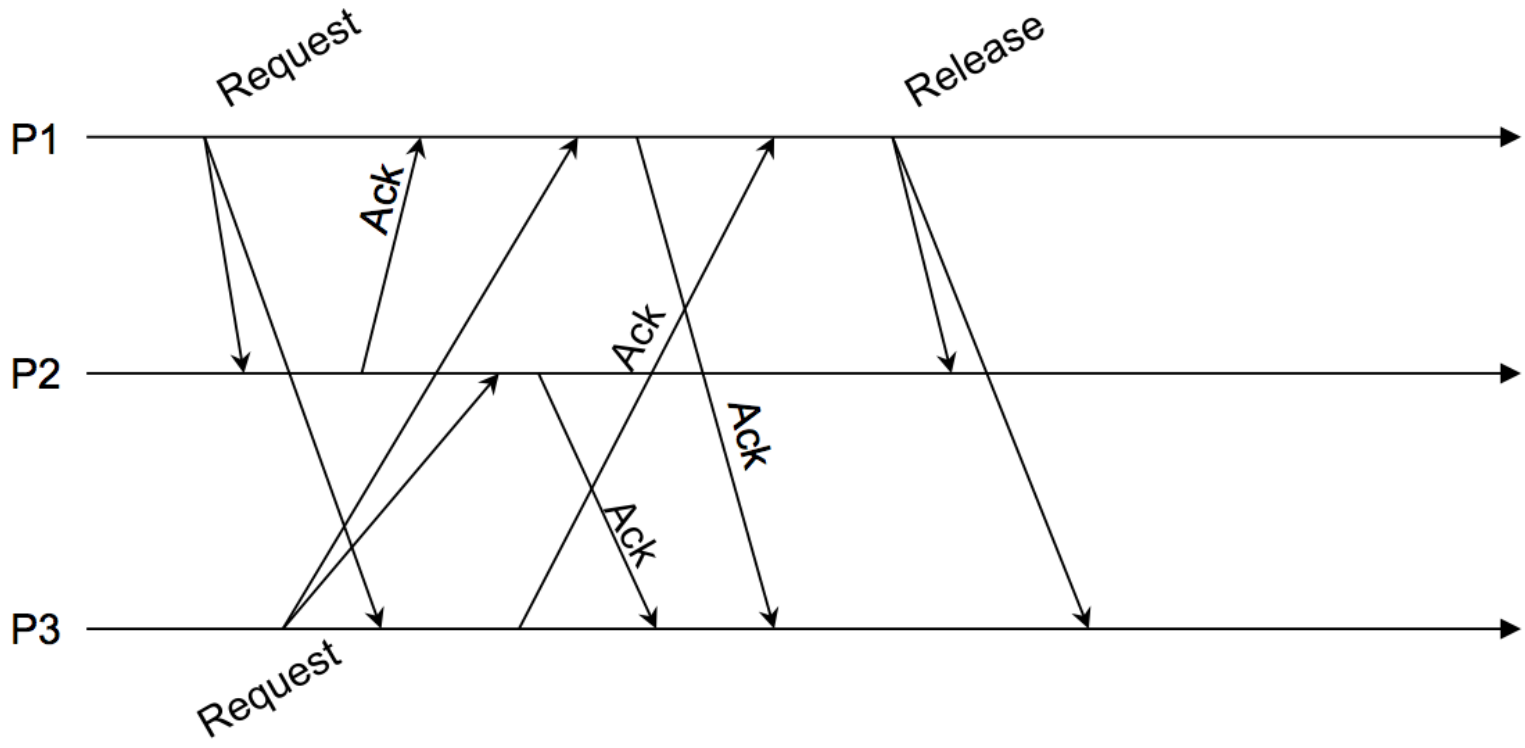
B1: Lamport-Zeit (V)



B.1.3) Einfügen einer Nachricht M, so dass

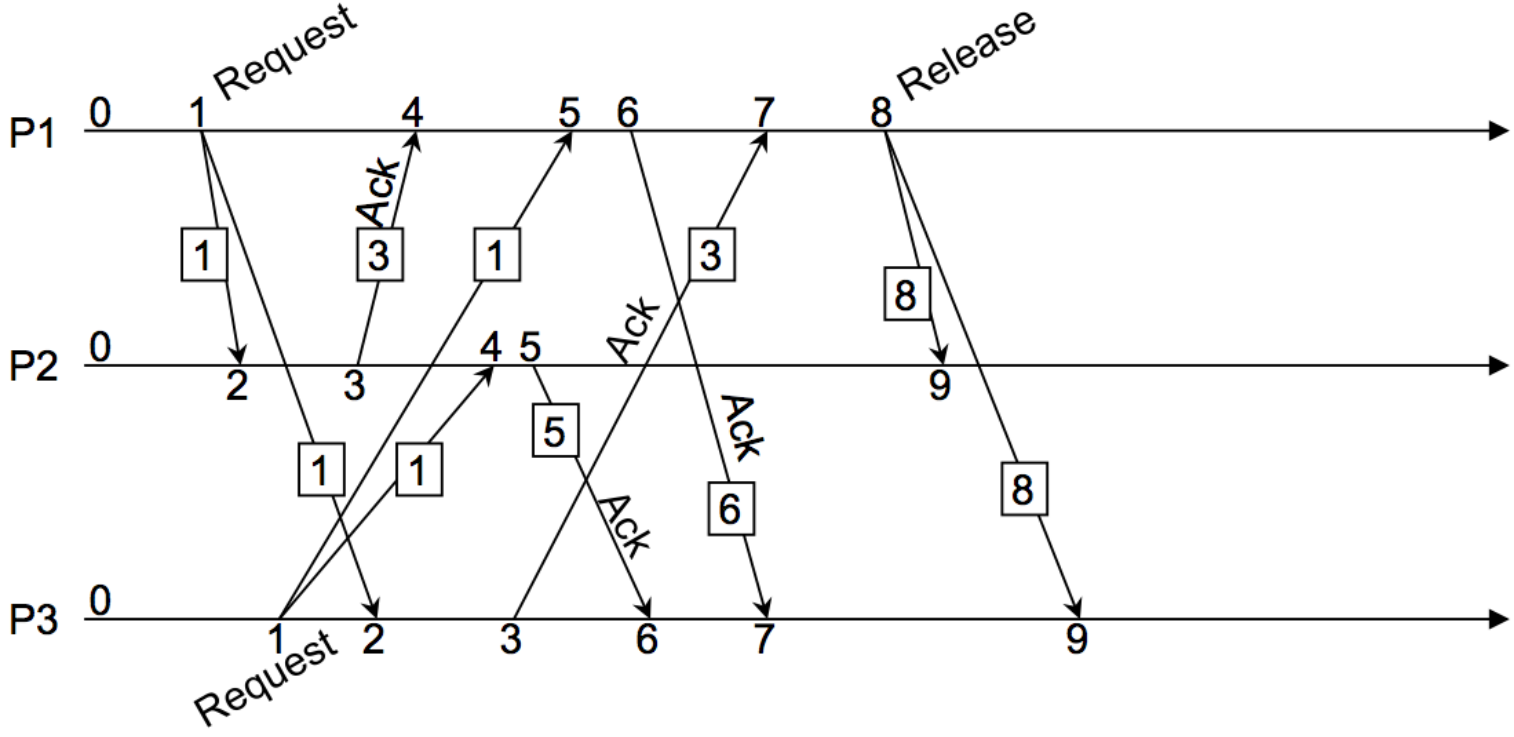
- M.send $\prec$ C.send $\wedge$ C(B.receive) $<$ C(M.receive)
- Sender: P2, Empfänger: P3

B2: Wechselseitiger Ausschluss mit Lamport-Zeit



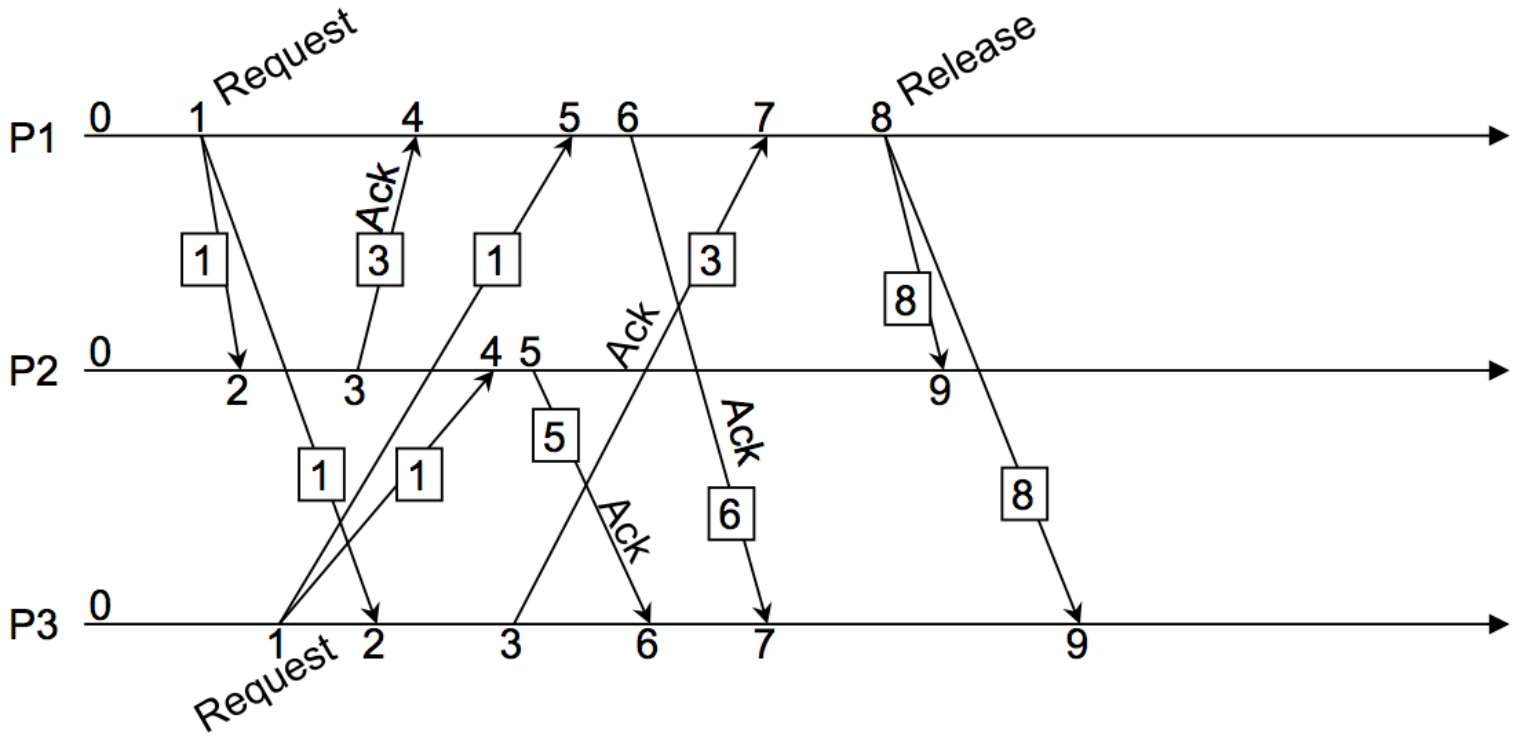
B.2.1 Sende- und Empfangszeitstempel für jedes Ereignis?

B2: Wechselseitiger Ausschluss mit Lamport-Zeit



B.2.1 Sende- und Empfangszeitstempel für jedes Ereignis?

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (II)



Die Prozesse wenden das aus der Vorlesung bekannte Verfahren zum wechselseitigen Ausschluss an, das Lamport-Zeit und verteilte Warteschlangen benutzt.

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (III)

Wiederholung: Algorithmus von Lamport (1978)

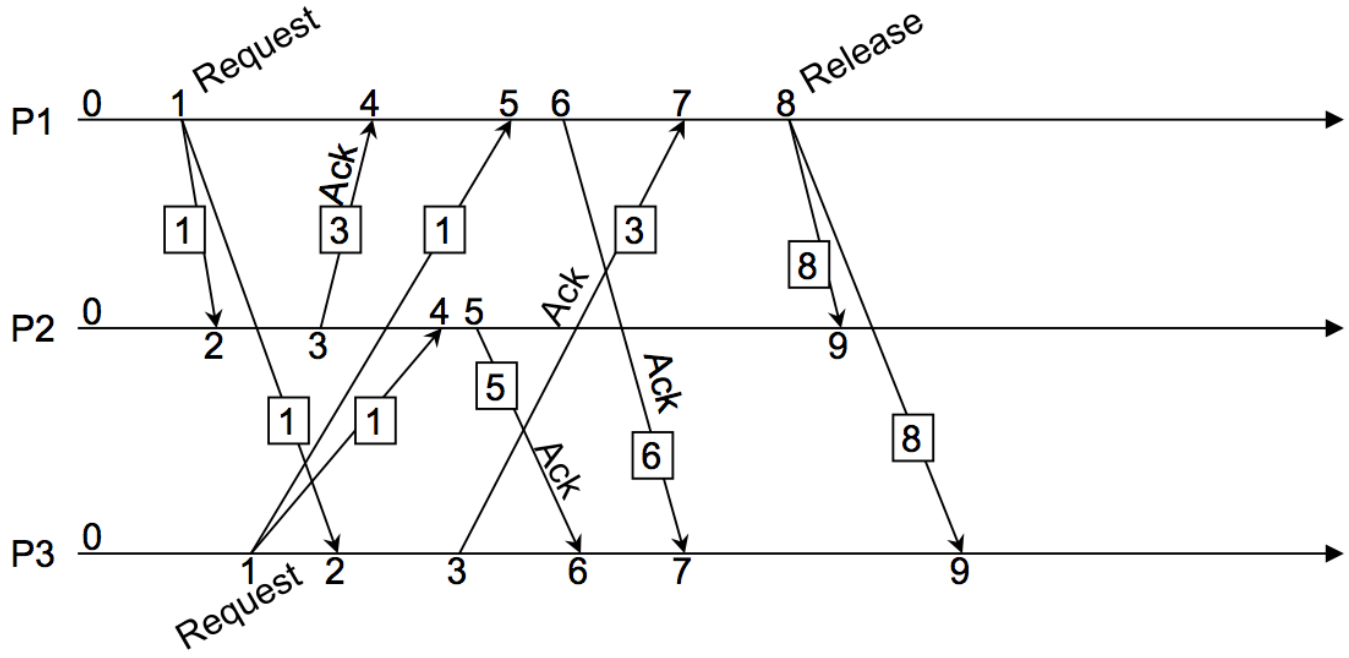
1. Bei „request" des Betriebsmittels: Request mit Zeitstempel und Absender an alle versenden und in eigene queue einfügen.
2. Bei Empfang einer request-Nachricht: Request in eigene queue einfügen, ack versenden.
3. Bei „release" des Betriebsmittels: Aus eigener queue entfernen, release-Nachricht an alle versenden.
4. Bei Empfang einer release-Nachricht: Zugehörigen request aus eigener queue entfernen.
5. Ein Prozess darf das Betriebsmittel benutzen, wenn gilt:
   - Eigener request ist frühester in seiner queue *und*
   - er hat bereits von jedem anderen Prozess (irgendeine) spätere Nachricht bekommen.

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (IV)

Wiederholung: Algorithmus von Lamport (1978)

- Problem: Lamport-Zeit ist nicht injektiv
- Erweiterung unter Einbeziehung einer Ordnung $\lhd$ auf Prozessen:
- $(C(e), P_i) < (C(e'), P_j)$
  $\Leftrightarrow C(e) < C(e') \vee C(e) = C(e') \wedge P_i \lhd P_j$
- Bei gleichem Zeitstempel ist das Ereignis auf dem „kleineren" Prozess das „frühere"
- Nun haben wir eine total geordnete Lamport-Zeit

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (V)



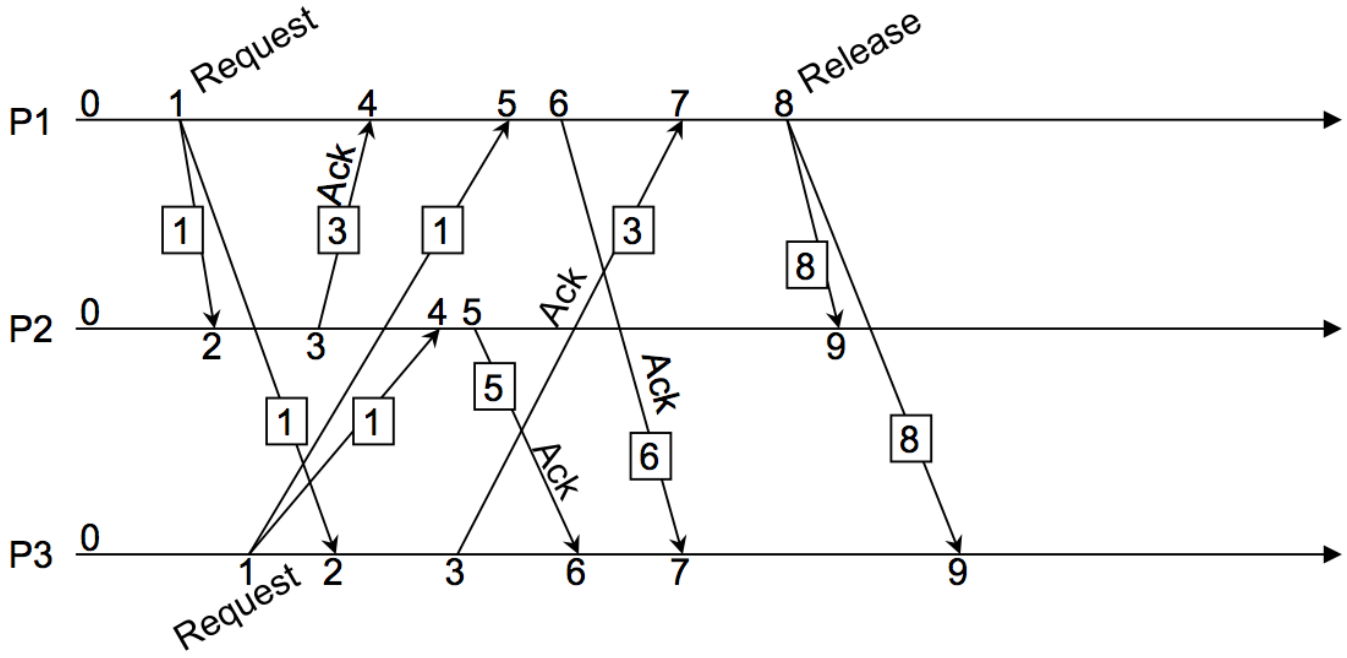
B.2.2. Geben Sie für die Prozesse 1 und 3 an, wie die Warteschlange des jeweiligen Prozesses nach jedem Sende- bzw. Empfangsereignis aussieht.

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (VI)

B.2.2. Beispiel Prozess 1:

1) Eigene Requestnachricht: Request(1,P1)
2) Ack von P2: Request(1,P1).
   (Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
3) Request von P3: Request(1,P1); Request(1,P3)
   (Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
4) Ack an P3: Request(1,P1); Request(1,P3)
   (Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3)
5) Ack von P3: Request(1,P1); Request(1,P3)
   (Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3,
   letzte Nachricht von P3 mit Zeitstempel 3)
6) Eigene Releasenachricht: Request(1,P3)
   (Zusätzlich: Letzte Nachricht von P2 mit Zeitstempel 3,
   letzte Nachricht von P3 mit Zeitstempel 3)

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (VII)

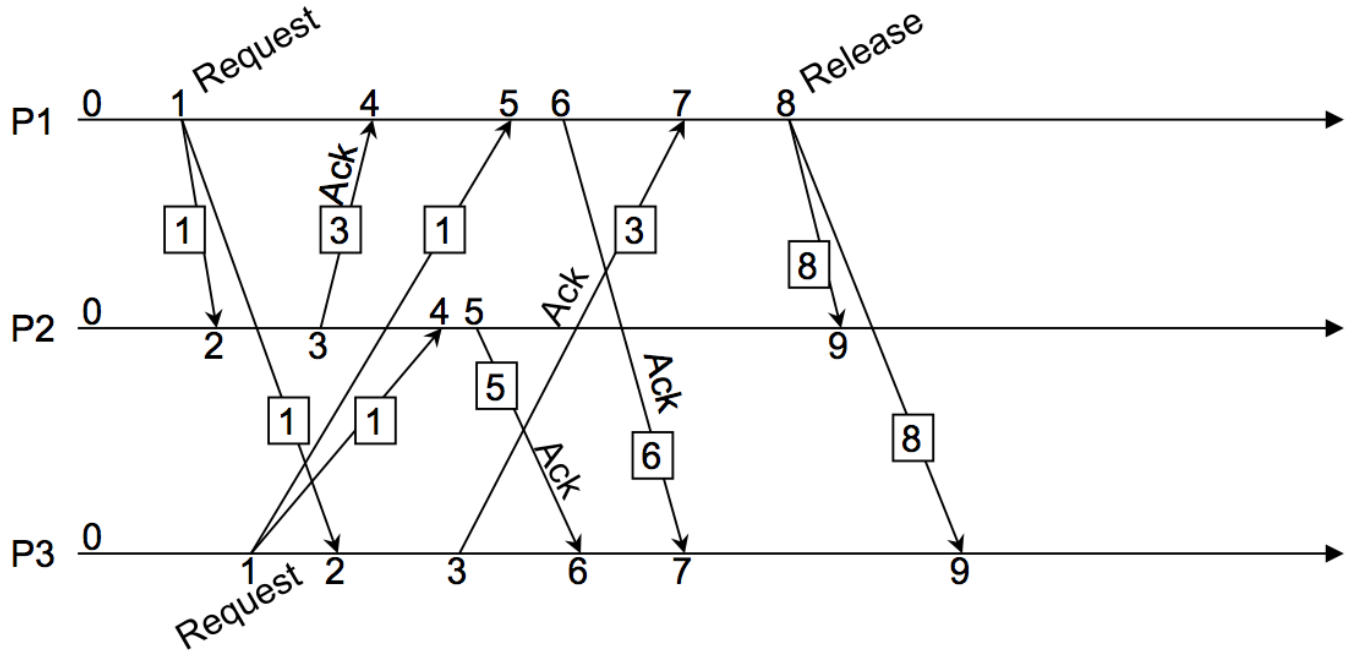


B.2.3. Sind beim Einreihen in die Warteschlange die Sende- oder die Empfangszeitstempel zu verwenden? Warum?

Die Sendezeitstempel, da sonst keine global-konsistente Sicht garantiert werden kann:

- Q1: Request(1,P1); Ack(4,P2); Request(5,P3); ...
- Q3: Request(1,P3); Request(2,P1); Ack(6,P2); ...

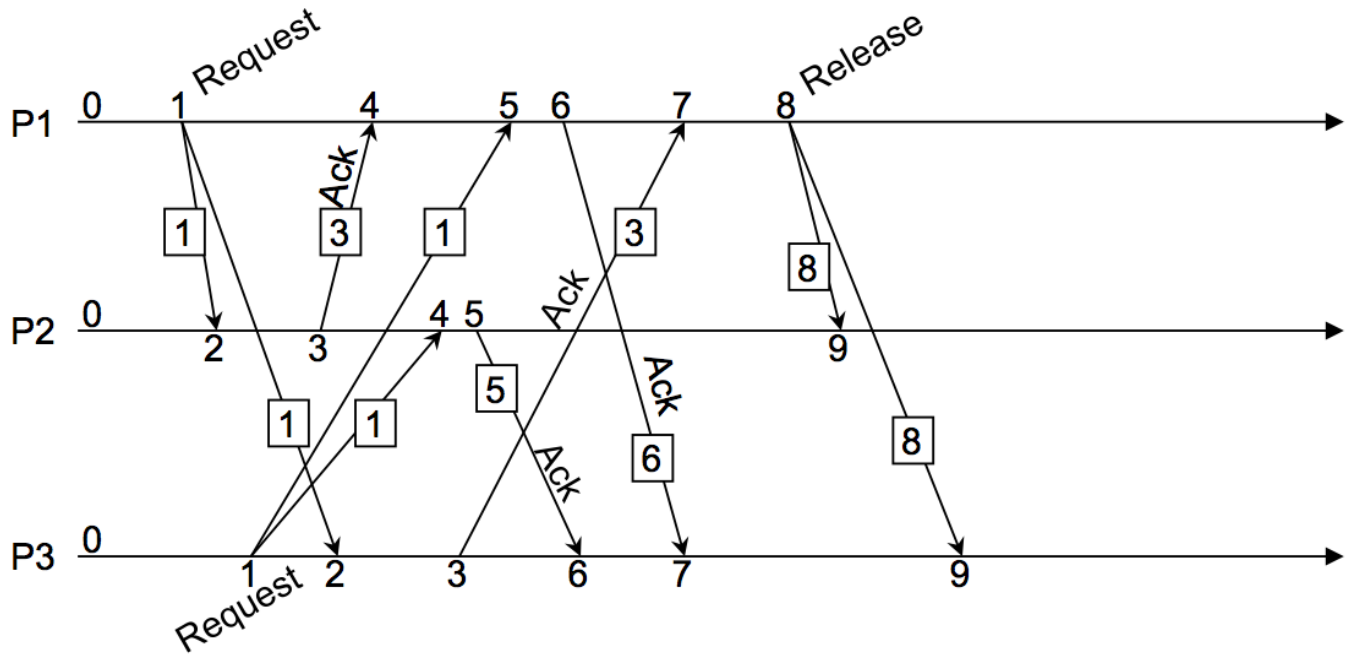
B2: Wechselseitiger Ausschluss mit Lamport-Zeit (VIII)



B.2.4. Welche Bedingung muss erfüllt sein, damit Prozess 1 auf die Ressource zugreifen kann?

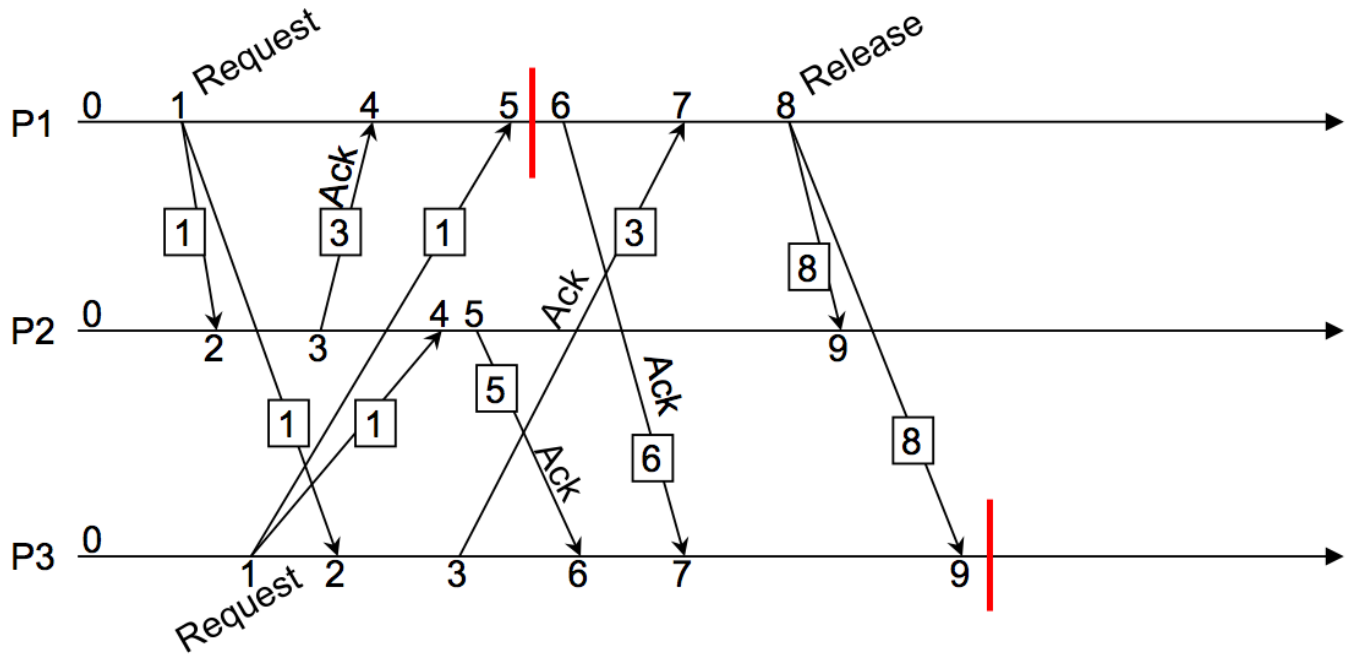
- Eigener Request ist der „früheste”
- „Spätere” Nachrichten von allen anderen Teilnehmern erhalten
- Achtung: Injektive Lamport-Zeit!

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (IX)



B.2.5. Markieren Sie den Zeitpunkt im Zeitdiagramm, zu dem Prozess 1 bzw. Prozess 3 auf die Ressource zugreifen kann.

B2: Wechselseitiger Ausschluss mit Lamport-Zeit (IX)



B.2.5. Markieren Sie den Zeitpunkt im Zeitdiagramm, zu dem Prozess 1 bzw. Prozess 3 auf die Ressource zugreifen kann.

B3: Namen

B.3.1 Was versteht man unter dem Binden und dem Auflösen von Namen? Nennen Sie ein Beispiel für einen solchen Dienst.

- Binden: Name wird einer Adresse zugewiesen.
- Auflösen: Adresse eines Namens wird „geholt".
- Beispiele: DNS und Telefonbuch.

B3: Namen (II)

B.3.2 Was ist der Unterschied zwischen iterativer und rekursiver Namensauflösung?

- Iterativ: Client bekommt Adresse des (nächsten) zuständigen Nameservers von seinem Nameserver mitgeteilt und muss diesen selbst kontaktieren.
- Rekursiv: Client kontaktiert nur seinen Nameserver, welcher den Namen auflöst, indem er gegebenenfalls andere Nameserver kontaktiert.

B.3.3. Unter welchen Voraussetzungen kann Caching bei der Auflösung von Namen effizienzsteigernd eingesetzt werden?

- Namensbindungen ändern sich selten
- Häufige Auflösungsanfragen zu gleichen Namen

B3: Namen (III)

B.3.4. Nehmen Sie an, die Abbildung von Namen auf Objektadressen wird in einem lokalen Cache eines Clients gespeichert. Ein bereits gebundener und im Cache aufgrund einer früheren Anfrage enthaltener Name wird nun an eine andere Adresse gebunden, das alte Objekt bleibt aber weiterhin aktiv.

a) Welches Problem tritt nun beim Client auf? Kann der Client dieses Problem erkennen?
   - Client kommuniziert mit dem ursprünglichem Objekt und erkennt das Problem nicht.

b) Tritt das Problem auch bei solchen Clients auf, die statt der Adresse den zuständigen Nameserver im Cache halten?
   - Nein.

B3: Namen (IV)

B.3.5. Zur Erhöhung von Effizienz und Fehlertoleranz werden Nameserver repliziert. Für welche Nameserver ist dies besonders relevant? Begründen Sie Ihre Antwort.

- Für Server, welche die höheren Hierarchieebenen abdecken (Root-Nameserver bzw. Nameserver der Top-Level-Domains).
- Es gibt hier sehr viele Anfragen. Ein Ausfall würde ganze Teilbereiche betreffen.

B4: Client/Server

B.4.1. Bei Web-basierten Diensten wird oft ein Bezeichner in Links codiert („URL rewriting"), um die aktuelle Transaktion zu identifizieren. Wie könnte ein Unbefugter eine laufende Transaktion „übernehmen" und was kann man gegen diese Gefahr tun?

- Unbefugter verwendet eine URL mit gültiger Transaktions-ID
- Timeouts erhöhen Sicherheit

Besser: Keine sicherheitsrelevanten Merkmale in der URL codieren.

B4: Client/Server (II)

B.4.2. Welches Problem entsteht bei einem zustandsbehafteten Server, wenn viele Clients abstürzen, bevor sie ihre Transaktionen beendet haben?

- Es werden Ressourcen auf dem Server blockiert, obwohl diese nicht mehr benötigt werden.

B.4.3. Erläutern Sie kurz ein paar Vorteile von zustandslosen gegenüber zustandsbehafteten Client/Server-Protokollen und umgekehrt.

- Vorteile zustandslos: Effizienz, Robustheit des Servers gegen eigenen Crash und Client-Crash
- Vorteile zustandsbehaftet: Sitzung kann über mehrere Interaktionen gehalten werden, potentielle Reduktion der übertragenen Datenmenge

B5: Middleware

B.5.1. Wie werden in CORBA verschiedene Programmiersprachen zur Implementierung der Anwendung unterstützt? Welchen Vorteil hat die Unterstützung mehrerer Sprachen?

- Mit Hilfe der Interface Description Language (IDL) können passende Client- und Server-Stubs erzeugt werden.
- Dadurch können heterogene Systeme miteinander kooperieren (wichtig u.a. bei Legacy-Systemen).

B.5.2. Welche Funktionen übernimmt ein Stub in CORBA?

- Verpacken und Entpacken der Parameter (Marshalling)
- Abschicken bzw. Entgegennehmen von Methodenaufruf-Mitteilungen über den ORB-Core.

B5: Middleware (II)

B.5.3. Was ist der ORB? Wo wird er ausgeführt?

- Der Object Request Broker ist das Rückgrat einer CORBA-Implementierung. Er dient als Vermittlungsschicht zwischen verschiedenen CORBA-Objekten, d.h. Weiterleitung von Methodenaufrufen, etc.
- Der ORB bildet eine Kommunikationsschicht und ist logisch über alle Teilnehmer verteilt.

B.5.4. Warum wird CORBA heutzutage nicht mehr weiterentwickelt?

- Zu weitreichende Anforderungen, fehlende Unterstützung durch Microsoft, aufkommende Konkurrenzsysteme, die zum Teil besser an die neuen Anforderungen angepasst sind, etc.

B5: Middleware (III)

B.5.5. Was versteht man unter Objekt-Serialisierung? Für was wird sie benötigt? Nennen Sie ebenfalls Beispiele, wie die Serialisierung in Middlewaresystemen realisiert wird.

- Unter Serialisierung versteht man die Konvertierung von Objekten in Byteströme. (Deserialisierung: Rückkonvertierung in eine identische Kopie des Ausgangsobjekts.)
- Ermöglicht den Austausch von Objekten zwischen verschiedenen Komponenten eines verteilten Systems.
- Java RMI bietet das Interface "Serializable", wodurch Objekte per ObjectInputStream/ObjectOutputStream (de-)serialisiert werden können.
- Bei .NET können Klassen mit einem Attribut versehen werden und deren Instanzen so binär oder als XML übertragen werden.

B6: Jini

B.6.1. Erläutern Sie kurz die Funktion von Leases.

- Zeitlich befristeter Vertrag über einen Service zwischen zwei Parteien.

B.6.2. Eine Besonderheit von Jini ist das Ausnutzen der Mobilität von Java-Code. Welche Code-Teile werden übertragen und welche Möglichkeiten ergeben sich dadurch?

- Proxy-Objekte implementieren die Serviceschnittstelle. Sie werden bei der Registrierung des entsprechenden Services beim Lookup-Service hinterlegt und bei der Nutzung des Services zum Client übertragen.
- Dies ermöglicht die Verwendung spezieller Protokolle sowie „smart proxies": Zusätzliche Intelligenz in den stubs um z.B. Vorverarbeitung oder Datenkompression durchzuführen.

B7: Sicherheit

B.7.1. Auf welche Herausforderungen trifft man bei der Schlüsselverteilung in verteilten Systemen?

- "Offenheit" verteilter Systeme fördert Angriffe
- Heterogenität der Systeme sorgt für zusätzliche Schwachstellen
- Keine zentrale Sicherheitsautorität

B.7.2. Beschreiben Sie zwei mögliche Lösungsansätze zur Schlüsselverteilung.

- Schlüsselvergabe über einen sicheren Kanal oder per (aufwendigem) Public-Key-Verfahren
- Direkte Schlüsselvereinbarung per Sicherheitsprotokoll (z.B. Diffie-Hellman).

B7: Sicherheit (II)

B.7.3. Der Diffie-Hellman-Algorithmus

a) Wofür wird der Algorithmus verwendet?

- Der Diffie-Hellman-Algorithmus stellt ein kryptographisches Protokoll dar. Dieses dient zur Erstellung eines geheimen Schlüssels zwischen Kommunikationspartnern über einen unsicheren Kanal.

B7: Sicherheit (III)

B.7.3. Der Diffie-Hellman-Algorithmus (II)

b) Beschreiben Sie kurz das Verfahren.

- Zwei Kommunikationspartner (A und B) kennen beide eine (grosse) Primzahl p und eine Konstante c, mit $1 < c < p$. Sie nutzen eine Einwegfunktion $f(x) = c^x \bmod p$.
- A und B wählen je eine Zufallszahl a bzw. b, die geheim gehalten werden muss.
- Aus den gegebenen Werten berechnen die Kommunikationspartner $\alpha = c^a \bmod p$ bzw. $\beta = c^b \bmod p$.
- A sendet α an B und B sendet β an A.
- Jetzt können A und B jeweils den gemeinsamen, geheimen Schlüssel berechnen: $G_A = \beta^a \bmod p$ und $G_B = \alpha^b \bmod p$.
- Da $(c^b \bmod p)^a \bmod p = (c^a \bmod p)^b \bmod p$ gilt, gilt auch $G_A = G_B$.

B7: Sicherheit (IV)

B.7.3. Der Diffie-Hellman-Algorithmus (III)

c) Was ist ein möglicher Angriff und wie könnte man sich dagegen verteidigen?

- Als „man in the middle" könnte man in den Kanal zwischen zwei Kommunikationspartnern eindringen und sich jeweils als Gegenstelle ausgeben. So werden für beide Teilstrecken eigene Schlüssel ausgehandelt und der Angreifer kann die Nachrichten transparent weiterleiten, sie dabei aber mitlesen und auch verändern.
- In der Vorlesung wurde in Verfahren vorgestellt, mit dem man diesen Angriff erkennen kann (→ Kap. Sicherheit, S.30).

B7: Sicherheit (V)

B.7.4. Wie funktioniert zertifikatsbasierte Authentifizierung? Von welchem weitverbreiteten System wird sie verwendet?

- Eine vertrauenswürdige Autorität signiert mittels asymmetrischer Verschlüsselung ein Zertifikat, welches die Identität einer Person oder Ressource zusammen mit dessen Public-Key enthält.
- Mittels eines Challenge-Response-Verfahrens überprüft die Gegenseite dann, ob die Person/Resource wirklich die ist, die sie vorgibt zu sein. Der Zertifikatsinhaber authentifiziert sich, indem er mit seinem Private-Key eine bestimmte Nachricht signiert, welche mit dem Public-Key aus dem Zertifikat validiert werden kann.
- SSL/TLS verwendet zertifikatsbasierte Authentifizierung.

B7: Sicherheit (VI)

B.7.5. Wenn One-Time-Pads ein perfektes Verschlüsselungssystem darstellen, warum werden diese dann heutzutage nicht global eingesetzt?

- One-Time-Pads sind nur unter der Annahme perfekt, dass die beteiligten Parteien bereits im Voraus über einen sicheren Kanal genügend Pads ausgetauscht haben.
- Die Pads müssen jeweils die gleiche Länge wie die Nachrichten haben und dürfen nur ein einziges Mal verwendet werden.
- Für Standardanwendungen daher kaum praktikabel.

B7: Sicherheit (VII)

B.7.6. Einwegfunktionen

f sei eine Einwegfunktion und x_1 ein initiales Passwort, aus dem eine Passwortkette erzeugt wird:

$$x_1 \rightarrow^f x_2 \rightarrow^f \ldots \rightarrow^f x_{n-1} \rightarrow^f x_n$$

a) Um die Passwörter zur Authentisierung nutzen zu können, muss x_n zunächst zum Server S übertragen werden. Welche der folgenden Anforderungen müssen erfüllt sein:
   i) Ein Angreifer darf nichts über x_n erfahren, die Übertragung muss also geheimnisbewahrend erfolgen. $\rightarrow$ Nicht erforderlich!
   ii) Es muss sichergestellt sein, dass x_n bei der Übertragung nicht verändert wird. $\rightarrow$ Erforderlich!

B7: Sicherheit (VII)

B.7.6. Einwegfunktionen (II)

b) Annahme: $n = 100$. Server S kennt x_{100}. Bei der zweiten Anmeldung verwendet Client C aus Versehen x_{89} (statt x_{98}). Welche Gefahr besteht, wenn dieser Wert von einem Angreifer abgehört wird und S den Anmeldeversuch einfach ignoriert, weil $f(x_{89}) \neq x_{99}$?

- Man setzt normalerweise voraus, dass die Einwegfunktion bekannt ist. Ein Angreifer könnte daher x_{89}, x_{90}, ..., x_{98} berechnen und einsetzen, d.h. er könnte sich bis zu 10-mal anmelden.

B7: Sicherheit (VIII)

B.7.7. Kann bei Kerberos ein verschlüsseltes Ticket Granting Ticket (TGT) von einem anderen Client verwendet werden, um vom Ticket Granting Service (TGS) ein Service Ticket (ST) anzufordern? Begründung!

- $TGT = \{Name, Client - ID, t, \Delta t, K, \dots\}_{KTGS}$
- Nein, da der Schlüssel K nur dem ursprünglichen Client bekannt ist. Um TGT zu verwenden, ist zusätzlich ein mit K verschlüsselter Authentizitätsnachweis erforderlich. Die Kenntnis von TGT allein reicht nicht aus.

B.7.8. Nennen Sie zwei Gründe, warum in Kerberos Key Distribution Center (KDC) und TGS getrennt sind.

- Skalierbarkeit
- konzeptionelle Trennung von Akkreditierung und Servicenutzung

Danke für Ihre Aufmerksamkeit