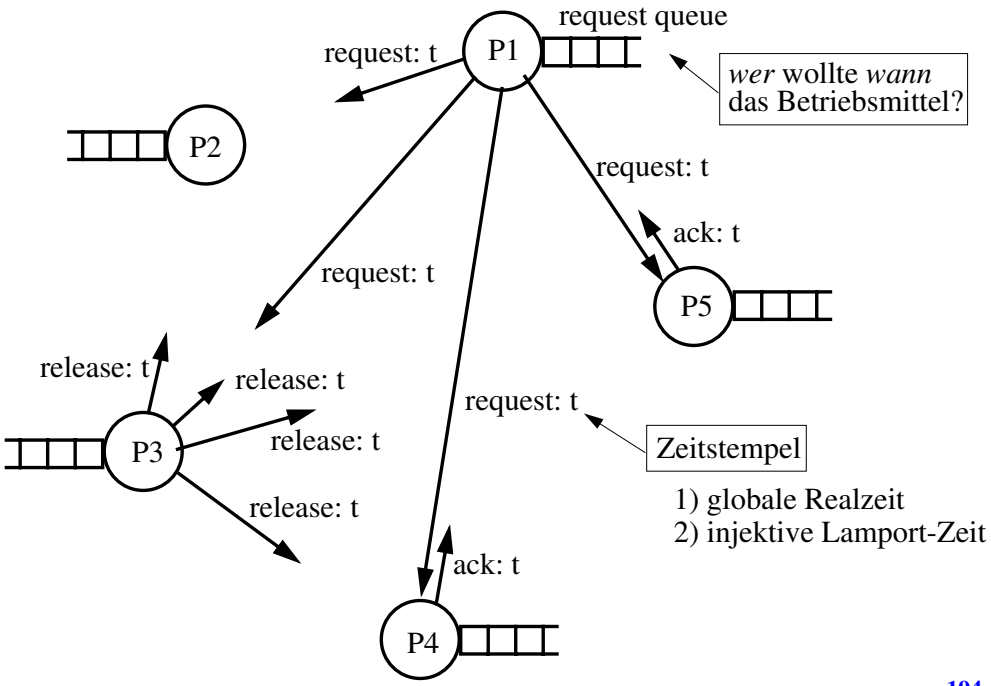
Anwendung logischer Zeit für den wechselseitigen Ausschluss

- Feste Anzahl von Prozessen
- Ein exklusives Betriebsmittel
- Synchronisierung mit request / release-*Nachrichten*
- Fairness: Jeder request wird schliesslich erfüllt

"request" / "release": --> vor Betreten / bei Verlassen des *kritischen Abschnittes*

Sehr schwache Fairnessanforderung

Idee: Replikation einer globalen request queue



Der Algorithmus (Lamport '78):

- Voraussetzung: Verlustfreie FIFO-Kanäle
- *Alle* Nachrichten tragen Zeitstempel (eindeutige!)
- Request- und release-Nachrichten an *alle* senden ← broadcast

1) Bei "request" des Betriebsmittels: Mit Zeitstempel request in die eigene queue und an alle versenden.

2) Bei Ankunft einer request-Nachricht: Request in eigene queue einfügen, ack versenden.
   - weiterer Nachr.typ
   - wieso notwendig?

3) Bei "release" des Betriebsmittels: aus eigener queue entfernen, release-Nachricht an alle versenden.

4) Bei Empfang einer release-Nachricht: Request aus eigener queue entfernen.

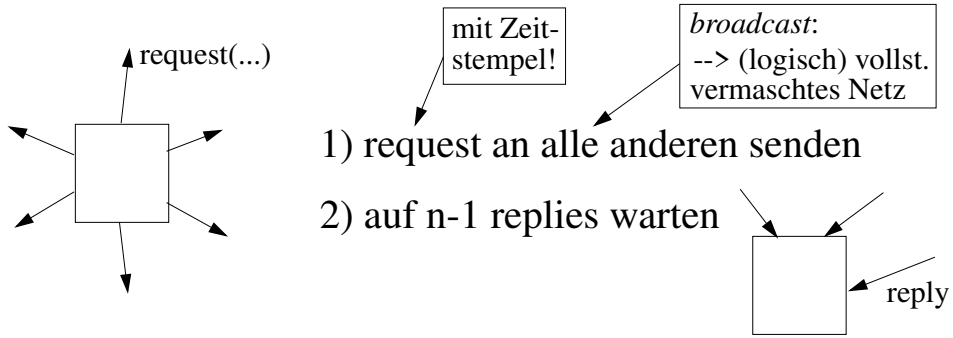
5) Ein Prozess darf das Betriebsmittel benutzen, wenn:
   - eigener request ist frühester in seiner queue und
   - hat bereits (irgendeine) spätere Nachricht von allen anderen Prozessen bekommen.

- Frühester request ist global eindeutig.
  ==> Bei 5): Sicher, dass kein früherer request mehr kommt (wieso?)
- Algorithmus ist nicht "optimal". ← Bessere Idee?
  ==> 3 (n-1) Nachrichten pro "request"

Denkübungen:
- Was könnte man bei Nachrichtenverlust tun? (--> Fehlertoleranz)
- Genauer (formal!) zu zeigen: safety, liveness (deadlockfrei), Fairness.
- Wo geht Uhrenbedingung / Kausaltreue der Lamport-Zeit ein?
- Ist FIFO notwendig? (Szenario hierfür?)

Ricart / Agrawala 1981: "An Optimal Algorithm for..."

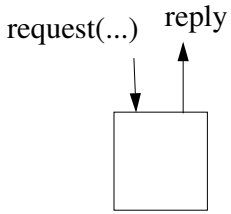
- 2(n-1) Nachrichten statt 3(n-1) beim Lamport-Verfahren:
  (*reply-Nachricht* übernimmt Rolle von release und ack)

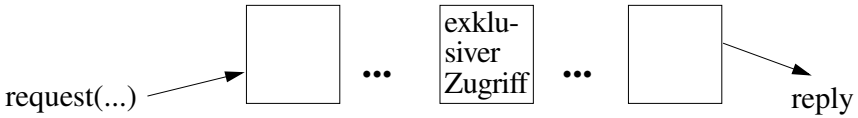


- Bei Eintreffen einer request-Nachricht:

  - reply sofort schicken, wenn nicht beworben oder der Sender "ältere Rechte" (logische Zeit!) hat,

  - ansonsten reply erst später schicken, nach Erfüllen des eigenen requests ("verzögern")





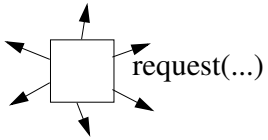
- Ältester Bewerber setzt sich durch!

Denkübungen:
- Safety, liveness: Argumente?
- Wie oft muss ein Prozess maximal nachgeben? (--> Fairness)
- Sind FIFO-Kanäle notwendig?
- Geht es nicht mit weniger Nachrichten? ("optimal")

Ricart / Agrawala 1983

1) Es gibt ein einziges Token; nur Besitzer darf kritischen Abschnitt betreten

2) Request-Nachricht (mit Zeitstempel) an alle senden



3) Token hat "Gedächtnis":

Prozess-nummer

Zeitstempel des letzten Besuchs (ggf. implizit 0, wenn Prozess noch unbekannt)

1	xxx
2	xxx
3	xxx
:	:
n	xxx

4) Jeder Prozess registriert Anforderungen *aller* anderen

5) Nach Verlassen des kritischen Abschnitts wird das Token an denjenigen Prozess geschickt, der am "längsten" wartet (Anforderung muss jünger als Zeitstempel des letzten Besuchs beim Prozess sein)

--> Eigenes Token für jedes "Betriebsmittel"

--> Nachrichtenkomplexität: n (n-1 für request, + 1 Token)
(bzw. 0, wenn inzwischen kein anderer wollte)

Fragen:
- Wie lange muss ein Prozess max. (auf Mitbewerber) warten?
- Liveness? Fairness?
- Ist n nicht die Mindestzahl von Nachrichten, die bei einem symmetrischen Verfahren benötigt werden?