Wechselseitiger Ausschluss

- Typischerweise exklusive Betriebsmittel
  - z.B. konkrete Betriebsmittel wie gemeinsamer Datenbus
  - oder abstrakte Betriebsmittel wie z.B. "Termin" in einem (verteilten) Terminkalendersystem
  - "kritischer Abschnitt" in einem (nebenläufigen) Programm

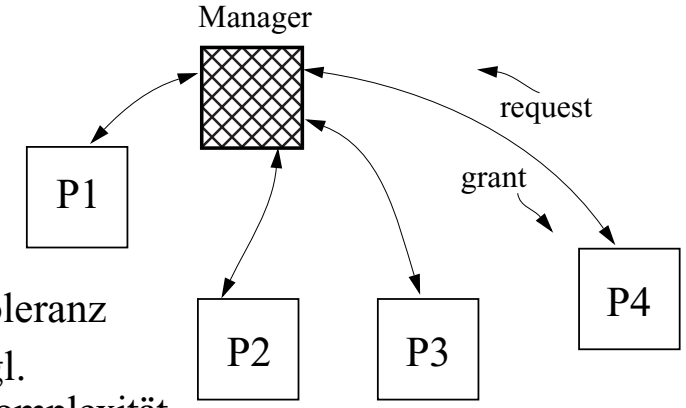
- Bei Einprozessormaschinen, shared memory etc.
  - Semaphore oder ähnliche Mechanismen

    ==> Betriebssystem- bzw. Concurrency-Theorie

    ==> interessiert uns hier nicht!

  - grundsätzlich interessant allerding: was sind die elementaren Basis-mechanismen, um wechselseitigen Ausschluss realisieren zu können?

- Nachrichtenbasierte Lösung, die uns nicht interessiert, da asymmetrisch ("zentralisiert"):

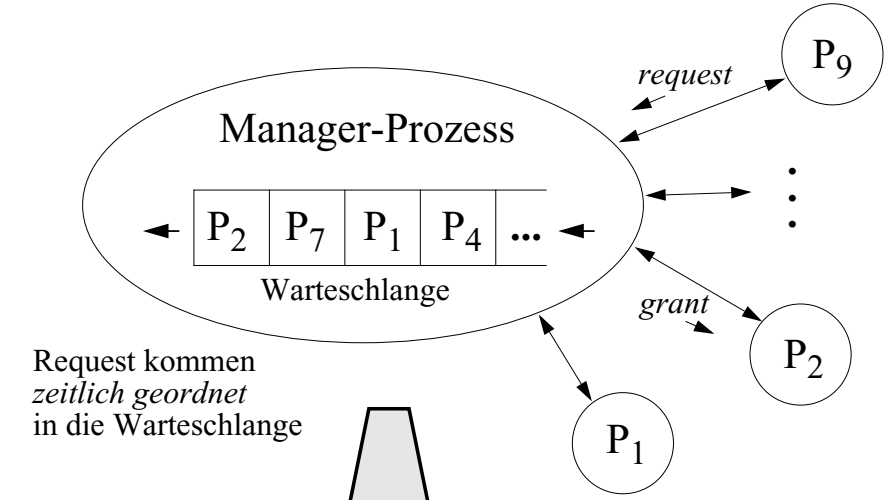


→ Engpass

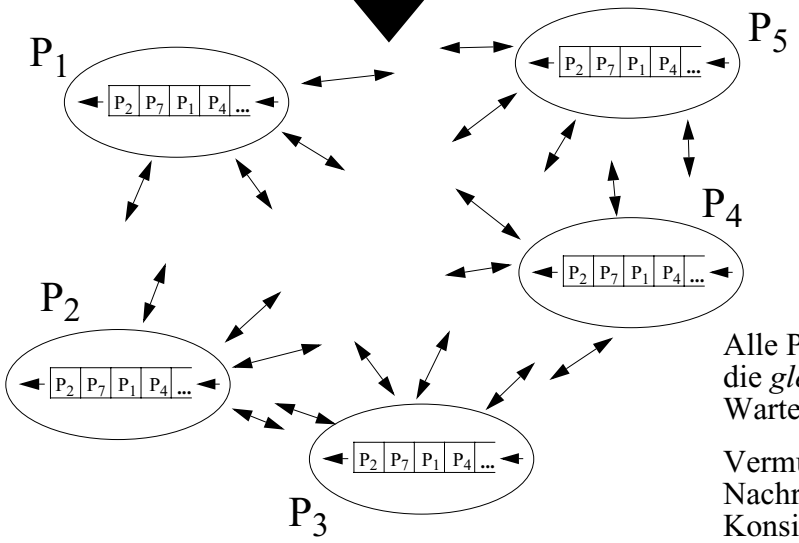
→ keine Fehlertoleranz

→ aber billig bzgl. Nachrichtenkomplexität

Replizierte Warteschlange?



Request kommen *zeitlich geordnet* in die Warteschlange



- Lässt sich mit "logischer Zeit" realisieren

Alle Prozesse sollen die *gleiche Sicht* der Warteschlange haben

Vermutlich viele Nachrichten, um Konsistenz zu gewähren; diese müssen ausserdem "geordnet" (=?) ankommen

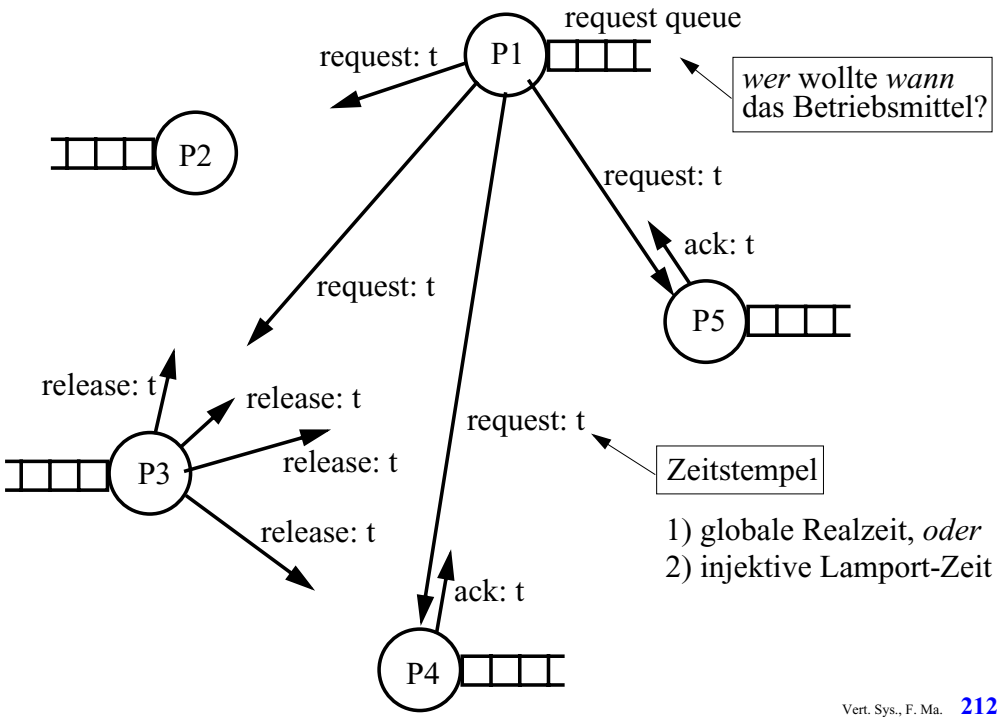
Anwendung logischer Zeit für den wechselseitigen Ausschluss

- 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



1) globale Realzeit, *oder*
2) injektive Lamport-Zeit

Der Algorithmus (Lamport '78):

- Voraussetzung: FIFO-Kommunikationskanäle
- *Alle* Nachrichten tragen (eindeutige!) Zeitstempel
- 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 Empfang 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?)

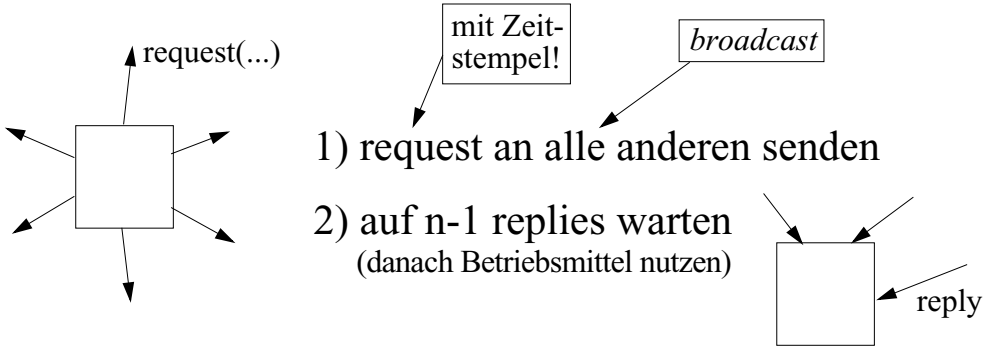
- 3 (n-1) Nachrichten pro "request"

Denkübungen:

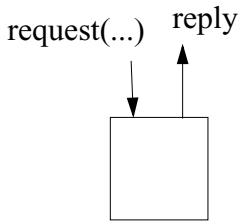
- wo geht Uhrenbedingung / Kausaltreue der Lamport-Zeit ein?
- sind FIFO-Kanäle wirklich notwendig? (Szenario hierfür?)
- bei Broadcast: welche Semantik? (FIFO, kausal,...?)
- was könnte man bei Nachrichtenverlust tun? (→ Fehlertoleranz)

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

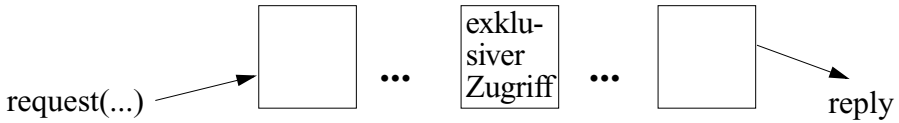
- 2(n-1) Nachrichten statt 3(n-1) beim Lamport-Verfahren:
  (*reply-Nachricht* übernimmt Rolle von release und ack)



- 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:
- Argumente für die Korrektheit? (Exklusivität, Deadlockfreiheit)
- Wie oft muss ein Prozess maximal nachgeben? (→ Fairness)
- Sind FIFO-Kanäle notwendig?
- Geht es wirklich nicht mit weniger Nachrichten? ("Optimal"?)