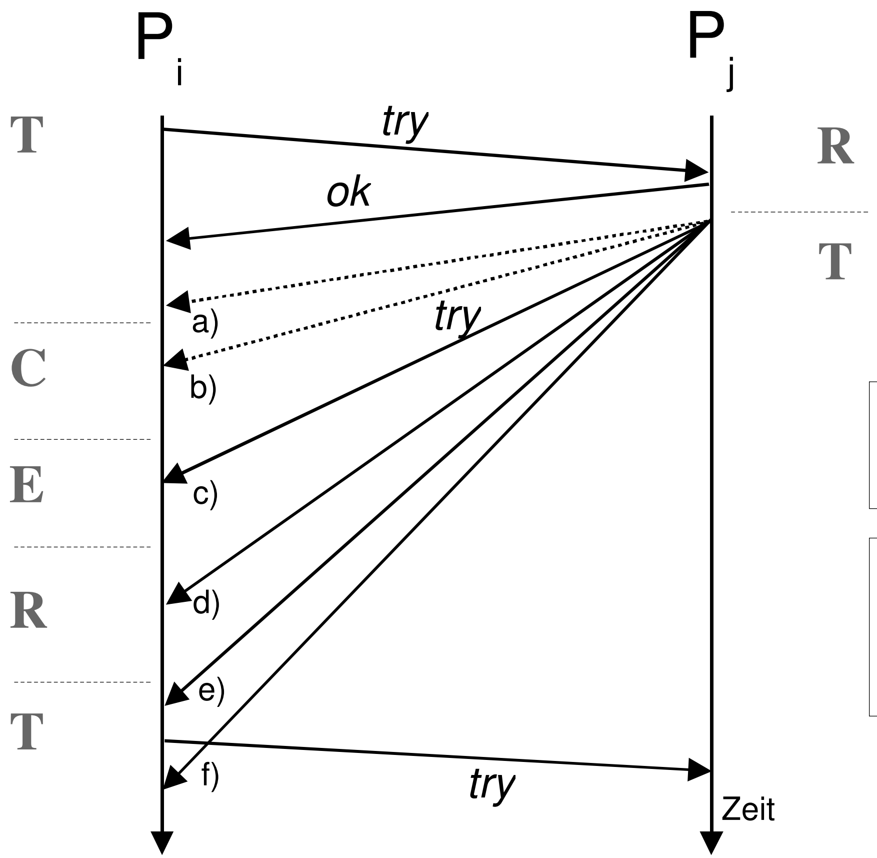
Mutual Exclusion
RicartAgrawalaME (I)

- ähnlich wie *LogicalTimeME*
- Antwort auf *try*: *ok*-Nachricht, evtl. verzögert
- Bedingung für *crit*: alle *ok*-Nachrichten erhalten

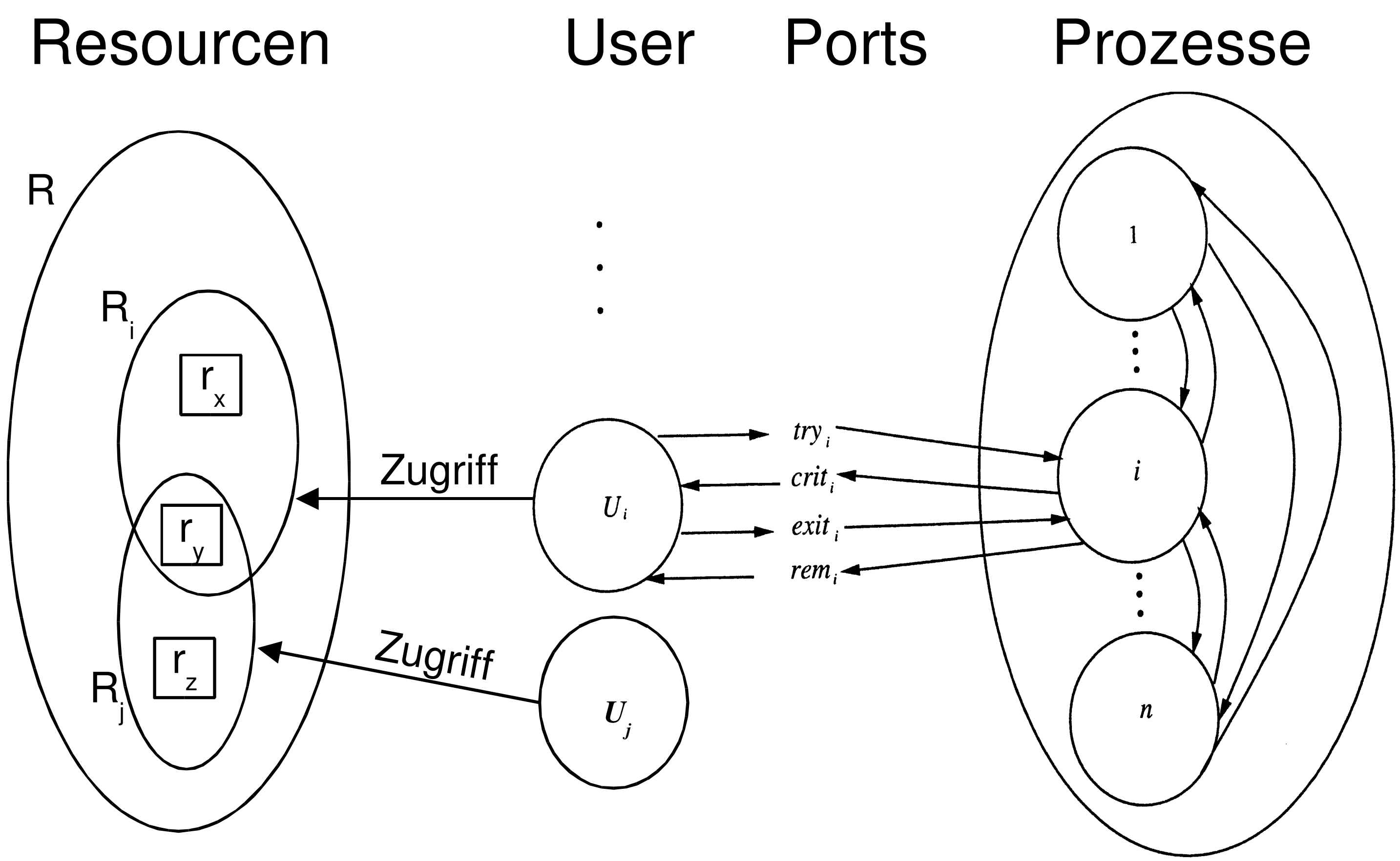
Mutual Exclusion
RicartAgrawalaME (II)
Regeln



c),d),e),f)	: P_i antwortet mit *ok*

a), b)	: P_i antwortet erst wenn er **C** verlässt

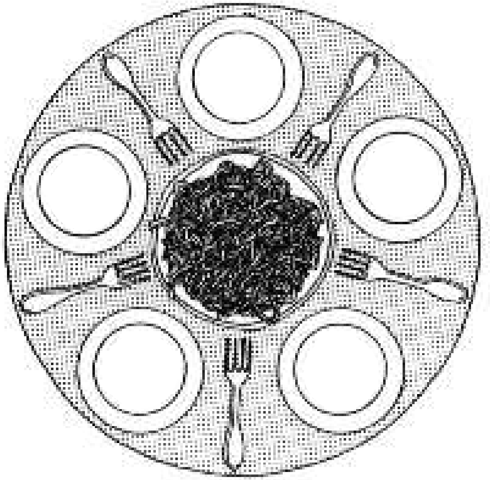
General Resource Allocation



General Resource Allocation
Korrektheitsbedingungen

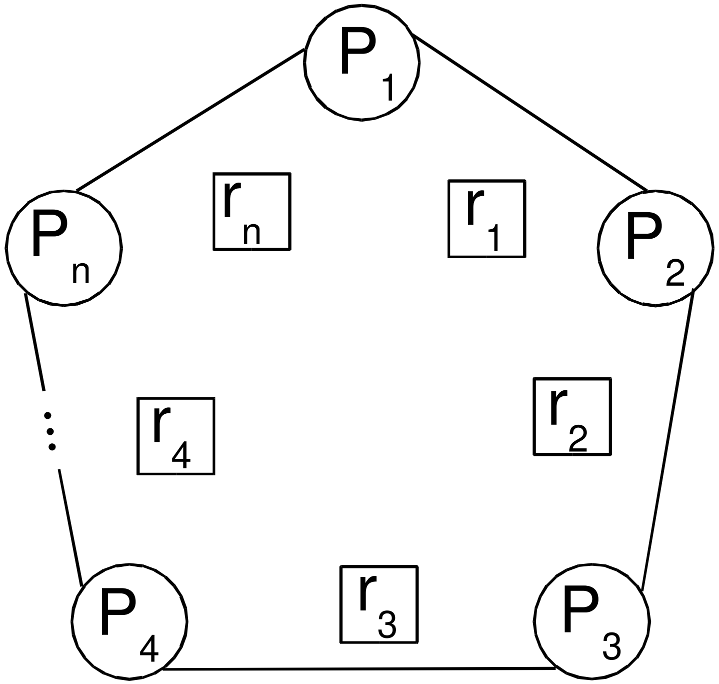
- Well-Formedness
- Exclusion
    - Menge der User in C nie Menge in $\mathcal{E}$
- Progress
- Independent Progress
- Lockout-Freedom

General Resource Allocation
Dining Philosophers (I)



- n Philosophen (*User*) sitzen um Tisch
- zwischen zwei benachbarten liegt eine Gabel (*Resource*)
- verbringt Zeit mit Denken (**R**)
- wird hungrig (betritt **T**)
- benutzt beide Gabeln (betritt **C**)
- legt beide Gabeln zurück (betritt **E**)
- setzt mit dem Denken fort (betritt **R**)

General Resource Allocation
Dining Philosophers (II)



- P_i: Prozess für User U_i
- Resourcen: $R=\{r_1,\ldots,r_n\}$

Exclusion Specification:

$$\mathcal{E} = \{\, E \subseteq \{\, i, i+1 \,\} : 1 \leq i \leq n \,\}$$

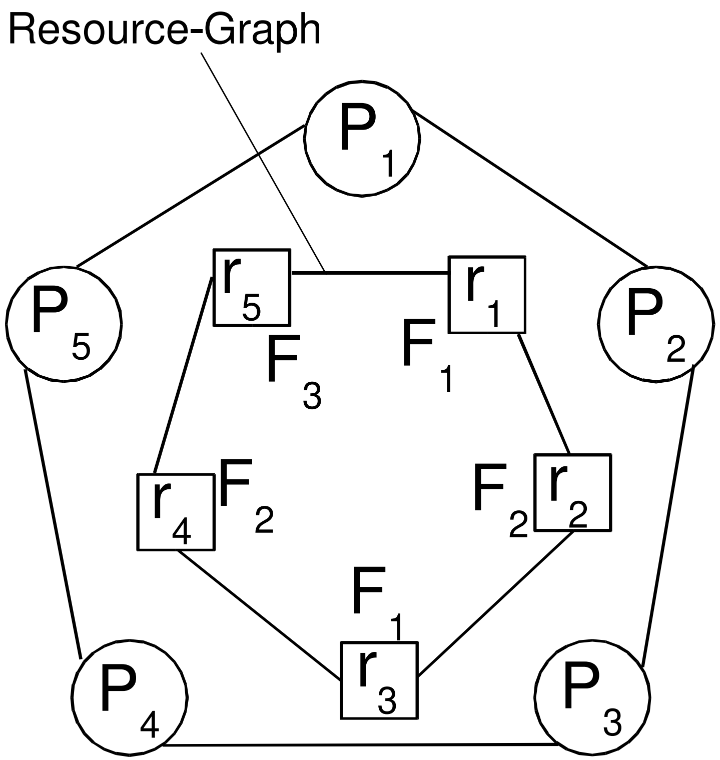
General Resource Allocation
Coloring Algorithmus (I)

- jeder Prozess P_i verwaltet $R^i \subseteq R$
- P_i sammelt benötigte Resourcen in Reihenfolge ermittelt durch *Färben*
- P_i sendet *try* an Prozess, der die Resource verwaltet und wartet auf Antwort
- $crit_i$ wenn alle Antworten erhalten
- $exit_i$: Nachricht an alle Resource-Prozesse

General Resource Allocation
Coloring Algorithmus (II)
Färben des Resource-Graphen

- Konstruktion des Resource-Graphen
  - Knoten $\triangleq$ Resourcen
  - Knoten verbunden $\Leftrightarrow$ ein Prozess benutzt beide Resourcen
- Färben des Resource-Graphen
  - benachbarte Knoten $\rightarrow$ verschiedene Farben
  - Anzahl verschiedener Farben minimieren
- Resourcen nach Farben sortieren

General Resource Allocation
Coloring Algorithmus (III)
Beispiel



- Reihenfolge für Farben festlegen:

$$\boxed{F_1 < F_2 < F_3}$$

$\Rightarrow$ *partielle* Reihenfolge für Resourcen:

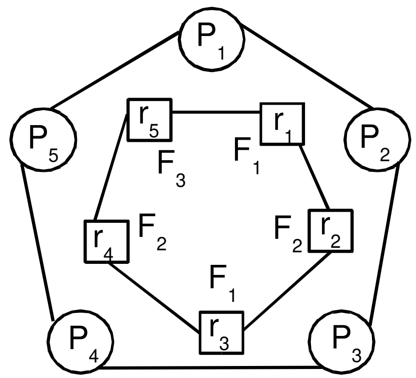
$r_i < r_j \Leftrightarrow$ (Farbe von r_i) < (Farbe von r_j)

$$\boxed{r_1 < r_2,\ r_3 < r_2,\ r_3 < r_4,\ r_4 < r_5,\ r_1 < r_5}$$

$\Rightarrow$ *totale* Reihenfolge für Resourcen eines Prozesses

General Resource Allocation
Coloring Algorithmus (IV)
Wartekette

- Länge einer *Wartekette* $\leq$ Anzahl der Farben
- Beispiel:
  - P_1 besitzt r_1 und r_5
  - P_5 nimmt r_4 und wartet auf r_5
  - P_4 nimmt r_3 und wartet auf r_4
  - P_3 wartet auf r_3



$\Rightarrow$ P_3 wartet auf P_4, der auf P_5 wartet, der auf P_1 wartet

- Wartet P_i auf eine Resource, die z.Zt. im Besitz von P_j ist, dann kann P_j nur auf eine Resource mit größerer Farbe warten

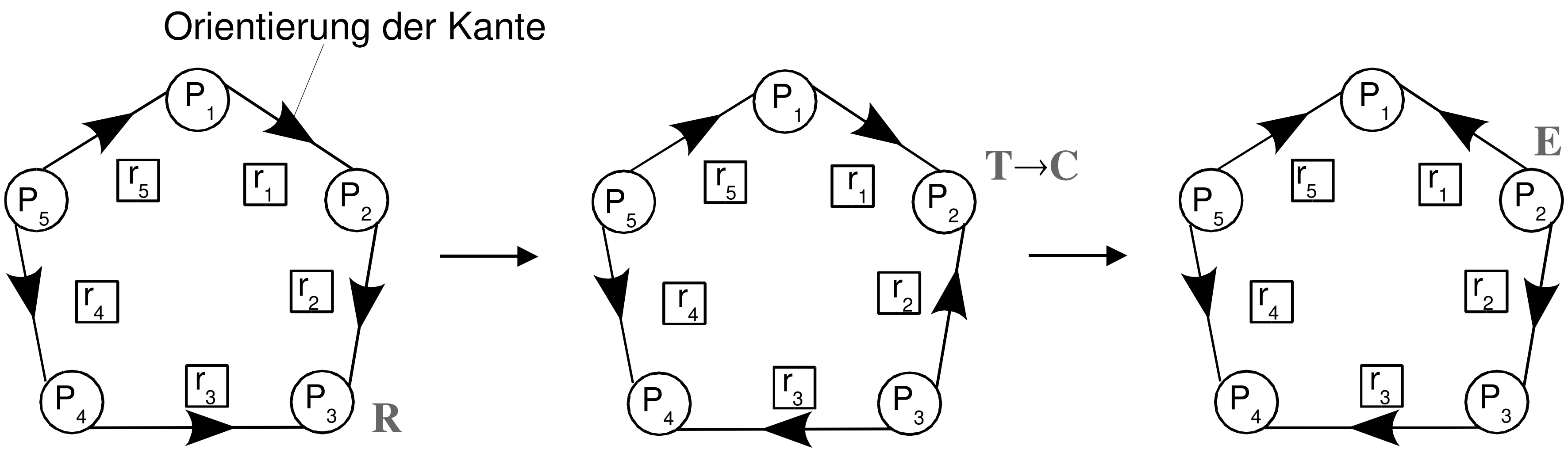
General Resource Allocation
RicartAgrawalaRA

- ähnlich wie *RicartAgrawalaME*
- *try multi*casten an konkurrierende Prozesse
- auf *ok*-Nachrichten (ggF. verzögert) warten

AcyclicDigraphRA (I)

- zur Vereinfachung:
  - jede Resource wird von zwei Prozessen geteilt
  - zwei Prozesse teilen sich höchstens eine Resource
- Graph konstruieren:
  - Knoten $\widehat{=}$ Prozesse
  - Kante zwischen konkurrierenden Prozessen
- Orientierung der Kanten
  - gespeichert in lokaler Variable beider Prozesse
  - *change*-Nachrichten entlang Kante ändern Variable

General Resource Allocation
AcyclicDigraphRA (II)
Beispiel



P_3 in **R** und alle Kanten zu sich hin gerichtet
⇒ Kanten von sich weg richten

P_2 in **T** und alle Kanten zu sich hin gerichtet
⇒ **C** betreten

P_2 verlässt **C**
⇒ Kanten von sich weg richten

General Resource Allocation
AcyclicDigraphRA (III)
Beweis

- eindeutige Orientierung einer Kante

⇒Exclusion

- Graph ist immer azyklisch
- *Höhe* h eines Knotens i: max. Pfadlänge
  - h erhöht sich nicht, bis h=0
  - Knoten mit h=0 richtet Kanten irgendwann von sich
  - Knoten mit h>0: irgendwann niedrigere Höhe

⇒Lockout-Freedom

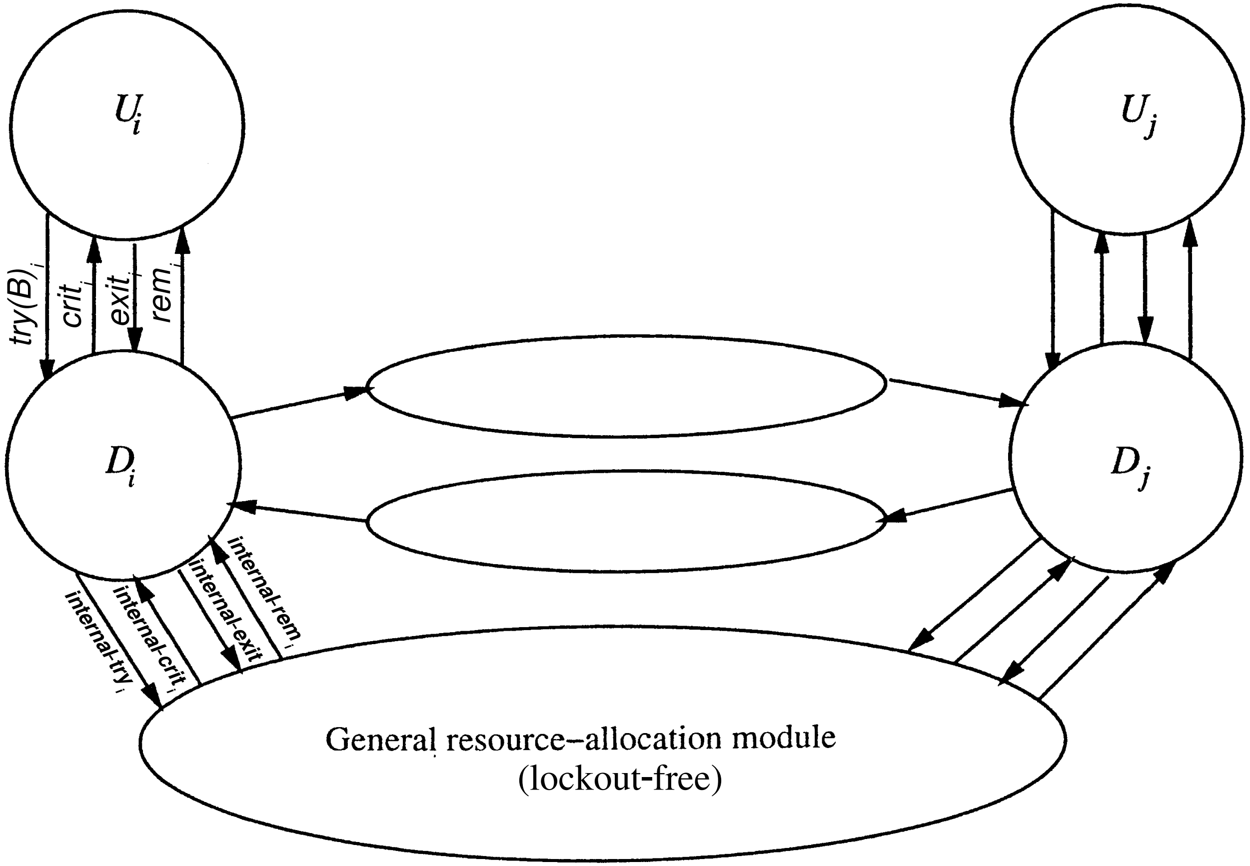
General Resource Allocation
Drinking Philosophers (I)

- Variation des RA-Problems
- User U_i kann variierende Teilmenge von R_i anfordern: $try(B)_i$, $B \subseteq R_i$
- Exclusion: Resourcen, die zur Zeit benötigt werden
- zur Vereinfachung:
  - jede Resource wird von zwei Usern geteilt
  - konkurrierende User sind durch Kante verbunden

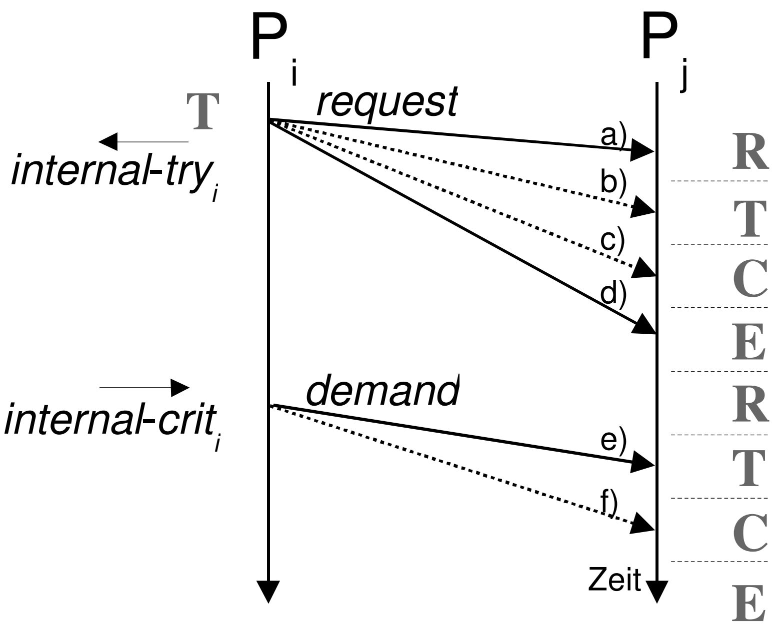
General Resource Allocation
Drinking Philosophers (II)
ModularDP

- Prozess D_i sendet Anfragen für Resourcen
  - D_j *mit r* in **T** oder **C**: verzögert Antwort bis **E**
  - sonst: D_j beantwortet die Anfrage sofort
- um gegenseitiges Blockieren zu verhindern:
  - RA-Modul → Priorität zwischen Prozessen
  - sobald D_i in **T**: *internal-try*$_i$
  - wenn *internal-crit*$_i$ → *demand*-Nachrichten

General Resource Allocation
Drinking Philosophers (III)
Architektur für ModularDP

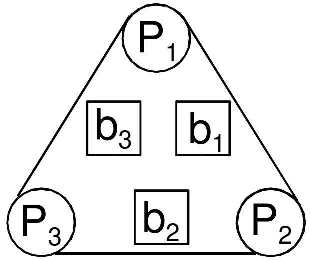


General Resource Allocation
Drinking Philosophers (IV)
ModularDP - Beispiel



a), d), e) : Anfrage wird sofort beantwortet

b), c), f) : Antwort wird verzögert bis **E**
falls Resource z.Zt. gebraucht wird



jeder Prozess
- besitzt rechte
- benötigt zusätzlich linke Resource