

Verteilte Systeme SS2002:

***Konsens und
Verlässlichkeit***

Marcel Waldvogel

Ordnung und Gleichzeitigkeit

■ *Einzelner Master*

- *Globales Locking*
- *Optimismus*
- *Vorteile?*

■ *Verteilte Master*

- *Zeitstempel*
- *Abstimmung*
- *Byzantinische Entscheidungsfindung*
- *Vorteile?*

Übersicht

- *Problemstellung*
 - *Fehler und -erkennung*
- *Lösungsmöglichkeiten*
 - *Ausschluss*
 - *Abstimmungen*
 - *Konsens*
- *Offene Probleme*
 - *Unzuverlässigkeit*

Fehler

■ Annahmen

- *Zuverlässige Kanäle*
 - *Endliche Auslieferungsverzögerung*
- *Unabhängigkeit*
- *Prozesse können terminieren (Crash)*

■ Weitere Möglichkeiten

- *Synchrone Kommunikation*
- *Hardware-Multicast*

■ Fehlertypen

- *Netzwerkpartitionierung*
- *Asymmetrie*
- *Intransitivität*

Fehlererkennung

- *Typischerweise unzuverlässig*
 - *Nicht Verdächtig, Verdächtig*
 - *Richtung des Irrtums?*
- *Zuverlässige Erkennung*
 - *Nicht Verdächtig, Fehlerhaft*
- *Timeouts*
- *Synchrone Systeme*

Gegenseitiger Ausschluss

- *Grundmechanismus*

- *Eigenschaften*

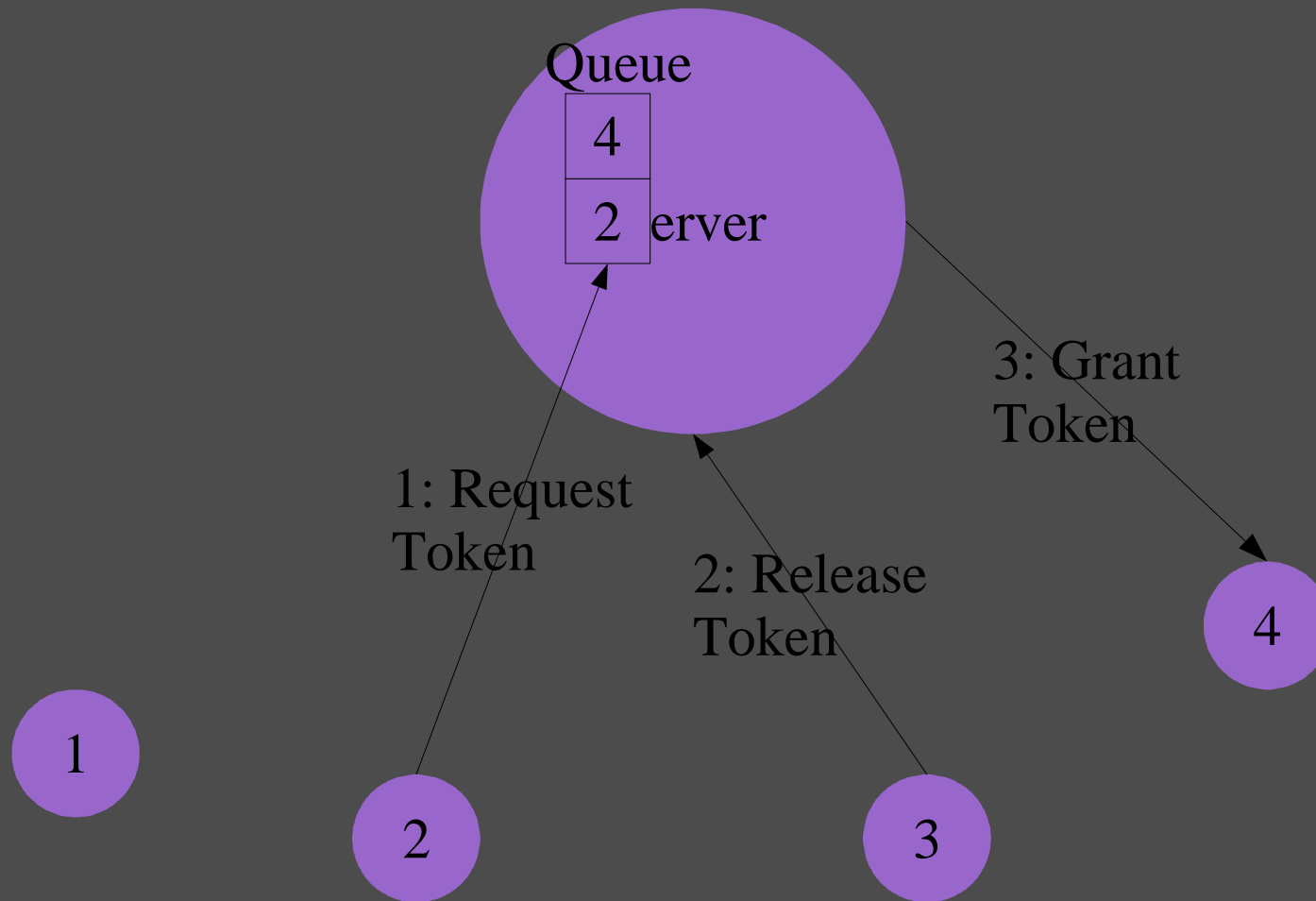
- 1. Sicherheit: Maximal ein Prozess in kritischem Bereich*

- 2. Lebendigkeit: Ein-/Austritt erfolgt nach endlicher Zeit*

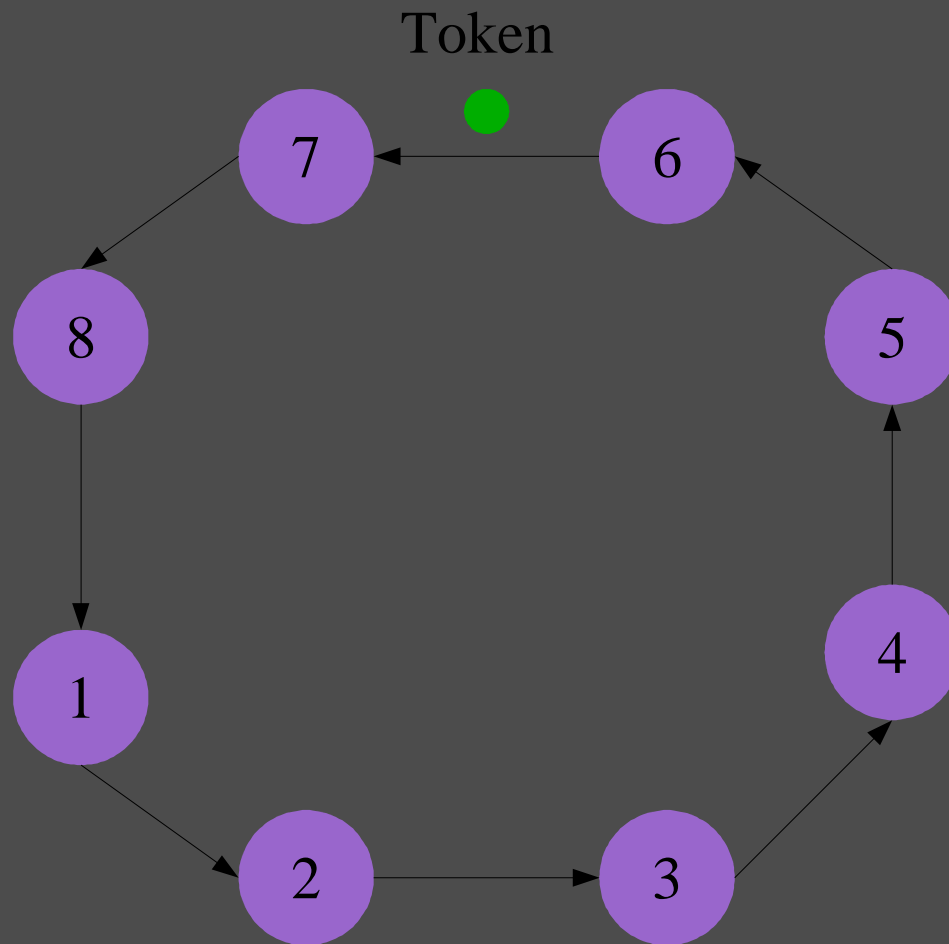
- 3. Ordnung (optional, fair): Anfragen in Happened-Before-Ordnung werden auch in dieser Reihenfolge bearbeitet*

- *Korollar: Kein Prozess erhält ein zweites Mal Zugriff auf den kritischen Bereich, bevor nicht alle länger anstehenden Wünsche befriedigt wurden*

Zentraler Server



Ring



Leistung

	Server	Ring	
Bandbreite	3	∞	[Msg/Entry]
Wartezeit	2	n	[Transmission]
Durchsatz	2	n	[Transmission]

Ausschluss mit Logischen Uhren

■ Initialisierung

- *status := FREI;*

■ Eintritt

- *status := ICH_WILL;*
- *Multicast Anfrage an alle;*
- *T := logische Zeit der Anfrage;*
- *Warte auf alle Antworten;*
- *status := BESETZT;*

■ Erhalt Anfrage $\langle T_i, p_i \rangle$ bei p_j

- *Falls Anfrage besser, beantworte sie; ansonsten stelle sie in Warteschlange.*

■ Austritt

- *status := FREI;*
- *Sende Antwort an alle in Warteschlange;*

Abstimmung zum Ausschluss

■ Erkenntnis

- *Nicht alle müssen zustimmen, es reicht, wenn jeder eine Teilmenge anfragt, und alle möglichen Teilmengen sich in mindestens einem Element überlappen*

■ Problem

- *Deadlock*

■ Lösung

- *Komplex*

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Fehlertoleranz

■ *Auswirkungen von*

- *Verlust einer Nachricht*
- *Absturz eines Prozesses*

■ *Umgehung des Problems?*

- *Verfügbarkeit eines zuverlässigen Fehlerdetektors*
- *Rekonstruktion des Systemzustandes vor Absturz?*

Wahlen

- *Zuverlässig auch bei Abstürzen*
- *Voraussetzungen*
 - *Sicherheit: Wenn ein Prozess einen anderen gewählt hat, sind sich alle einer Meinung*
 - *Lebendigkeit: Alle (funktionierenden) Prozesse nehmen teil und bilden sich irgendwann eine Meinung*
- *Möglichkeiten*
 - *Ring (siehe Occam)*
 - *Tyrann*

Bully Algorithm

- *Synchrones System*
 - *Timeout*
- *Erkennen von Abstürzen*
- *Kenntnis aller höher priorisierten Prozesse*
- *Nachrichten*
 - *Wahl ("Der König ist tot!")*
 - *Antwort (ich lebe noch)*
 - *Koordinator ("Es lebe der König!")*
- *Effizienz?*

Multicast

- *Implosion*
- *Offene und geschlossene Gruppen*
- *Zuverlässigkeit*
- *Ordnung*
 - *Sinn?*

Zuverlässiger Multicast

■ Kriterien

- *At-most-once*
- *Auslieferung in endlicher Zeit*
- *Alle oder keiner (für korrekte Prozesse)*

■ Senden

- *Zuverlässiges Multi-Unicast an alle (inklusive Sender);*

■ Empfang bei p

- *Wenn Nachricht kein Duplikat ist:*
 - *Falls p nicht originaler Sender ist, dann schicke Meldung mit zuverlässigem Multi-Unicast an alle (inklusive Sender);*
- *Liefere Nachricht lokal aus;*

Ordnung bei Multicast

- *Unabhängig von Zuverlässigkeit*
- *FIFO-Ordnung*
 - *Meldungen eines Senders werden nicht umsortiert*
 - *Sequenznummern*
- *Kausale Ordnung*
 - *Happened-Before bleibt erhalten*
 - *Vektorzeiten*
- *Totale Ordnung*
 - *Jeder Prozess erhält jede Nachricht in derselben Reihenfolge*
 - *Empfangsreihenfolge unabhängig von Sendereihenfolge*
 - *Zentraler Server für Sequenznummern*
 - *Hardware-Multicast (Vorsicht bei Ethernet!)*

Verteilte Totale Ordnung

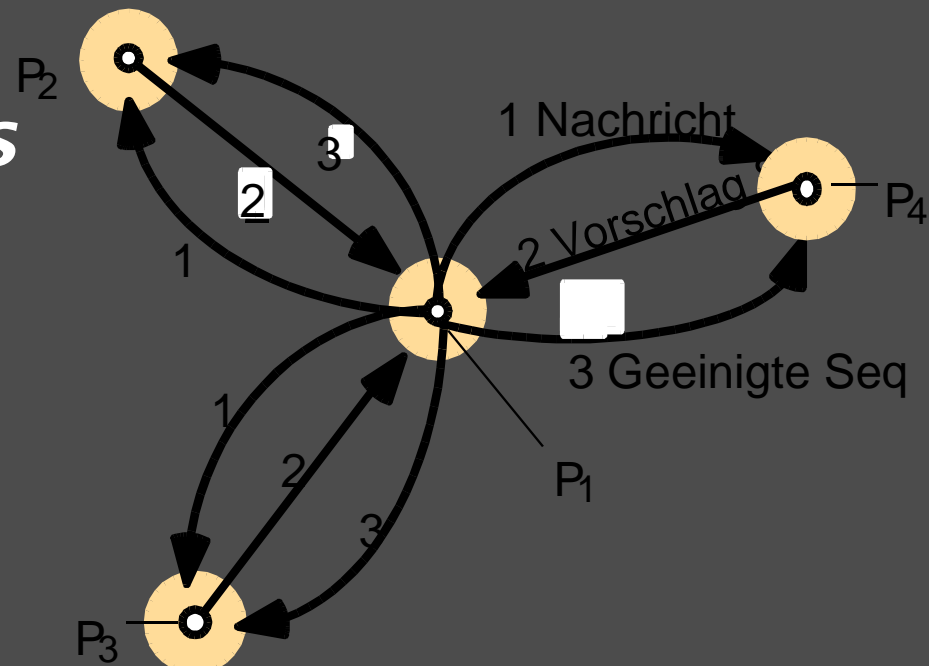
- Nachricht mit zuverlässigem Multi-Unicast verschicken
- Alle Antworten mit einer vorgeschlagenen Sequenznummer

– $P_q := \text{Max}(A_q, P_q) + 1$

- Grösste Antwort als Sequenznummer a senden

– $A_q := \text{Max}(A_q, a)$

- Ausliefern



Beispiel: Diskussionsgruppen

Total: Alle Nummern sind überall dieselben

Bulletin board: *os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

Kausale Ordnung:
Original vor Antworten
darauf

FIFO: Senderreihenfolge

Konsens

■ Frage

- *Wie einigt man sich auf einen Entscheid, wenn einiges schief läuft?*

■ Annahmen

- *Kommunikation zuverlässig*
- *Prozesse nicht*

■ Weitere Annahmen

- *Byzantinische Fehler möglich*

Konsens: Ablauf

- *Jeder Prozess ist unentschieden*
- *Jeder schlägt einen Wert vor*
- *Kommunikationsphase*
 - *Crash möglich*
- *Jeder Prozess entscheidet sich irgendwann für einen der vorgeschlagenen Werte*
- *Ziel: Alle korrekten Prozesse einigen sich innert nützlicher Frist auf dasselbe Resultat*

Byzantinische Entscheidungsfindung

- **Byzanz/Konstantinopel, 1453**
 - **Mehrere Generäle mit Truppen**
 - **Nur gleichzeitiger Angriff möglich**
 - **Widrige Umstände**
 - **Klassisch: Konspiration einiger Generäle**
 - **Alternative: Meldungsübertragung gestört**
- **Modern**
 - **Einigung mehrerer Rechner/Programme eines Verteilten Systems auf gemeinsames Resultat**
 - **Fehler im System (absichtlich und unabsichtlich)**

Byzantinische Generäle

■ *Problem*

- *4 Generäle*
- *Nicht alle loyal*
 - *Verräter geben falsche Informationen weiter*
 - *Verräter befolgen Befehle*
- *Direkte, perfekte Kommunikation*
- *Eine (korrekte) Entscheidung*

■ *Einfache Lösung*

- *Abstimmung*
 - *Jeder schickt seine Stimme an alle*
 - *Jeder errechnet "lokal" die Stimmenmehrheit*
 - *Verräter befolgen Ergebnis auch*
 - *Funktioniert nicht*

■ *Viele Varianten, komplexere Lösungen*

Anwendungen

- *Klassisch: Militärische Koordination*
- *Informatik: Fehlertoleranz*
 - *Flugzeugsteuerung*
 - *Zuverlässige Rechnersysteme*
 - *Verteilte Systeme*
 - *Nutzung von Rechnern unter unbekannter Administration*
 - *Peer-to-peer-Systeme*