Motivation: Verteilte Systeme

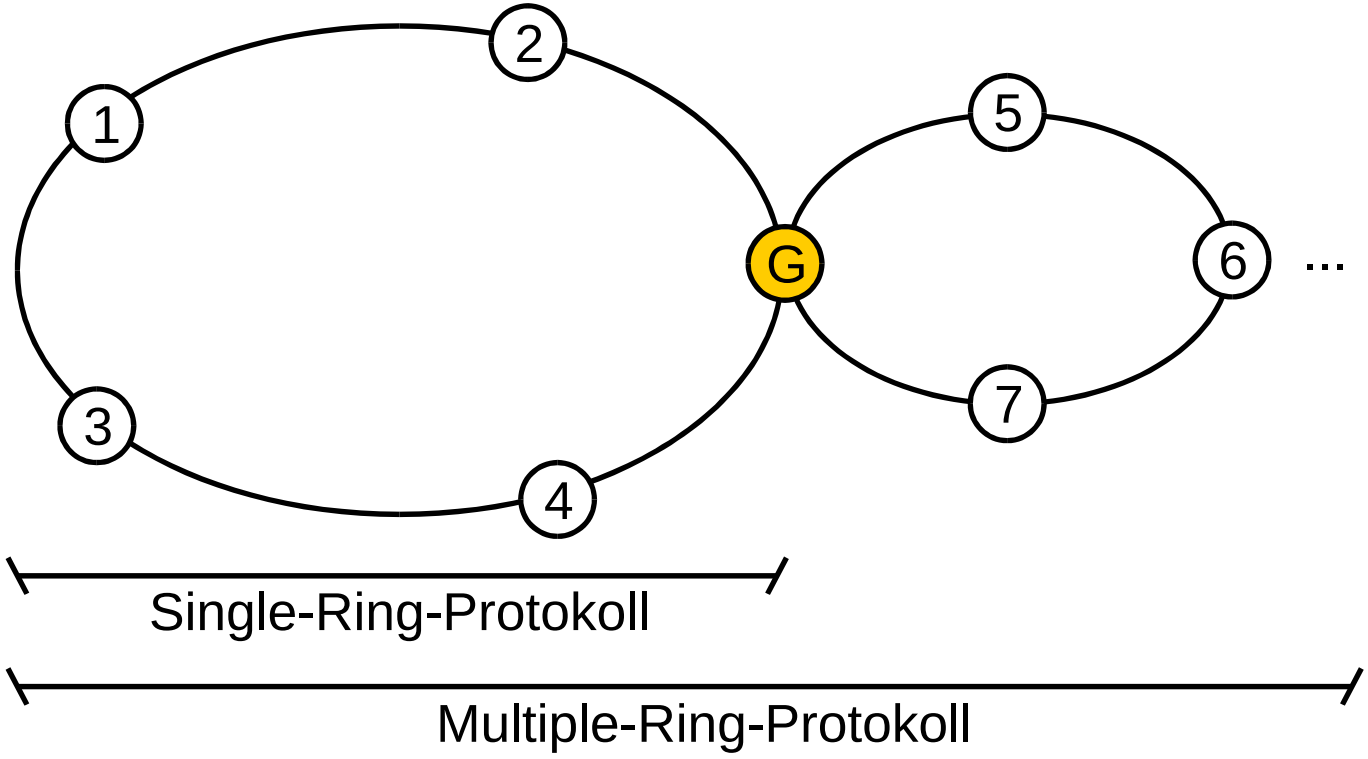
- Vorteile
  - geringe Kosten
  - hohe Verfügbarkeit
  - gute Performance
- Nachteile
  - Anwendungsentwicklung schwierig

Motivation: Verteilte Systeme (2)

- Anwendungsbeispiele
  - Prozesssteuerung
  - Datenbanksysteme
  - u.v.m.

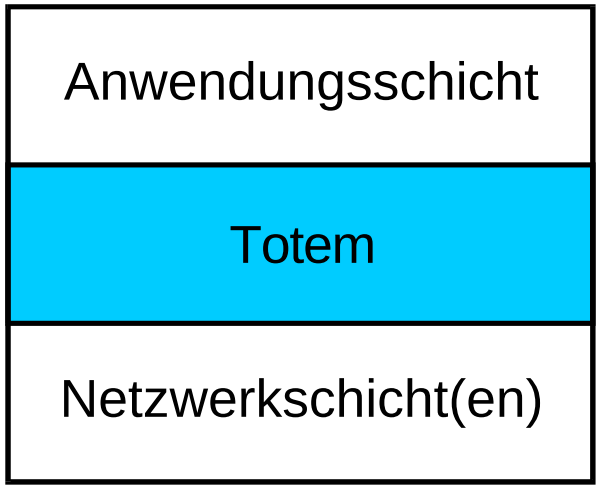
Totem: Umgebung

- Broadcast-Domänen
  - Prozessoren 1, 2, 3, ...
  - Gateway G



Totem: Umgebung (2)

- Protokollstack



Totem: Umgebung (3)

- Broadcast-Domänen
  - asynchrones Netzwerk
  - logischer Token-Ring
  - Token regelt den Zugriff auf das Kommunikationsmedium
  - ein Repräsentant in jedem Ring
- Nachrichten
  - werden immer an alle anderen gesendet (Broadcast)
  - Token: Punkt-zu-Punkt-Nachricht

Totem: Umgebung (4)

- Dienstmerkmale
  - zuverlässige Nachrichtenauslieferung
  - totale Ordnung
  - “agreed” oder “safe delivery”
  - Flusskontrolle
  - Fehlererkennung durch Timeouts

Totem: Umgebung (5)

- Begriffe
  - Konfiguration: Sicht auf das System
  - Mitgliedschaft: Menge von Prozessor-IDs
  - zu unterscheiden sind “empfangen”, “ausliefern” und “erzeugen” von Nachrichten

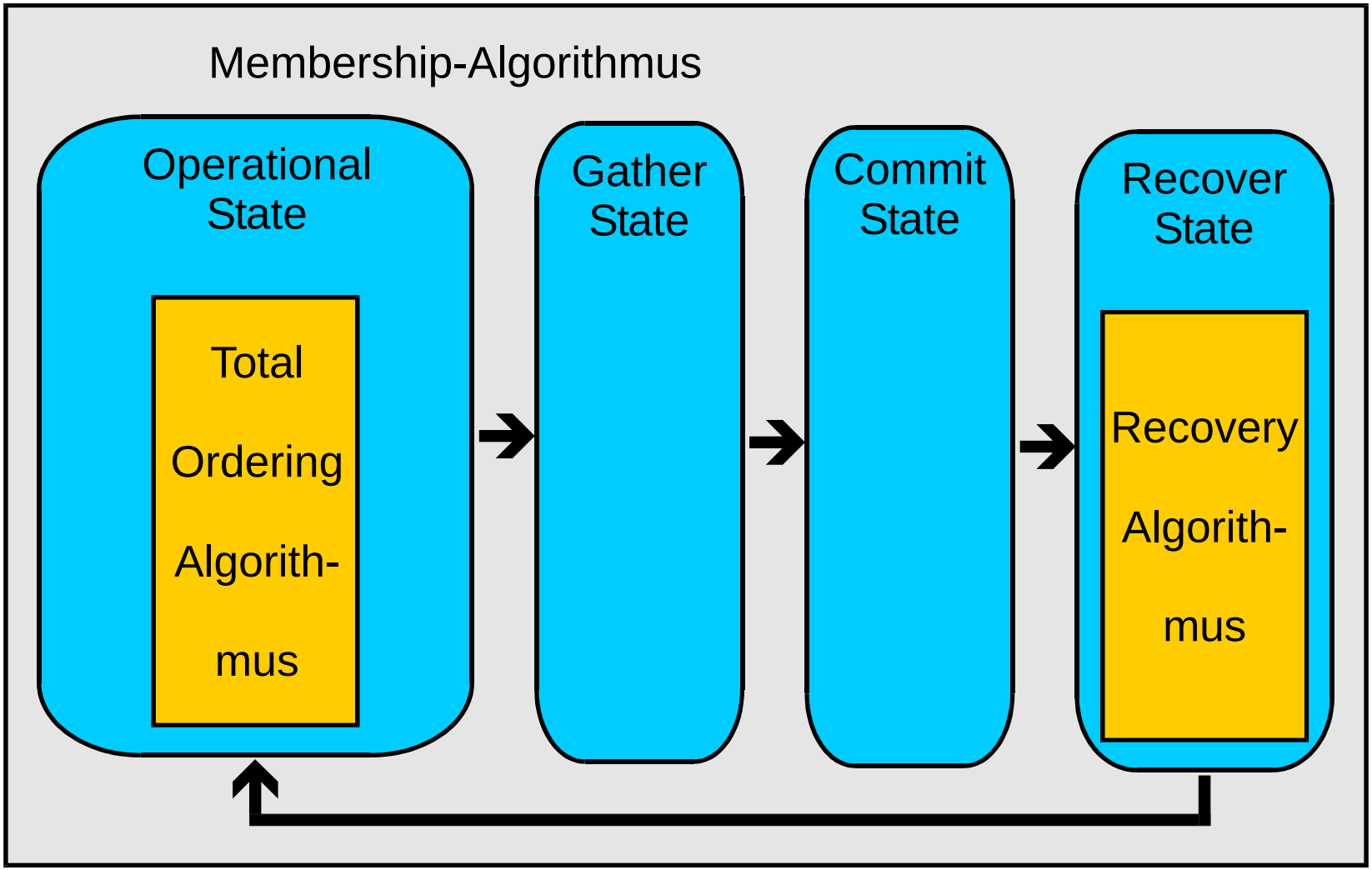
Totem: Umgebung (6)

- Nachrichtentypen
  - reguläre Nachricht
  - Configuration Change-Nachricht

Totem: Fehlermodell

- Nachrichtenverlust ist möglich
- Prozessoren können fehlerhaft arbeiten oder ausfallen
  - keine byzantinischen Fehler!
- Netzwerk kann partitioniert werden
- Verlust des Tokens führt zur Bildung eines neuen Rings (Membership-Algorithmus)

Totem: Übersicht



Totem: Total Ordering-Algorithmus

- Aufbau einer regulären Nachricht

sender_id	*ring_id*	*seq*	*conf_id*	*contents*

- Aufbau des regulären Tokens

type	*ring_id*	*token_seq*	*seq*	*aru*	*aru_id*	*rtr*

  - *aru*: all-received-up-to
  - *rtr*: retransmission request list

Totem: Total Ordering-Algorithmus (2)

- Lokal aufbewahrte Werte
  - *my_aru*
  - *my_token_seq*
  - *my_high_delivered*

Totem: Total Ordering-Algorithmus (3)

- Empfang des Tokens
  - Prüfen von *token_seq* (redundante Tokens)
  - angeforderte Neuübertragungen (*rtr*) broadcasten
  - Vergleich *aru* mit *my_aru*
  - Vergleich *aru_id* mit eigener ID
  - eventuell Neuübertragungen anfordern (in *rtr*, falls *seq* > *my_aru*)
  - Abbruch eines Token-Loss-Timeouts

Totem: Total Ordering-Algorithmus (4)

- Empfang einer Nachricht
  - Auslieferung in “agreed order”: alle Nachrichten mit kleinerer Sequenznummer wurden schon empfangen und ausgeliefert
  - Auslieferung in “safe order”: zusätzlich wurde das Token in zwei Runden mit *aru* größer oder gleich der Sequenznummer der Nachricht weitergegeben
  - Abbruch eines Token-Loss-Timeout

Totem: Membership-Algorithmus

- Vier Zustände
  - Operational State: Total Ordering-Algorithmus wird ausgeführt (s.o.)
  - Gather State: Einigung über die neue Ringmitgliedschaft
  - Commit State: Bestätigung der Mitgliedschaft und Sammeln von Informationen für den Recover State
  - Recover State: Neuübertragung alter Nachrichten

Totem: Membership-Algorithmus (2)

- Auslöser zur Bildung eines neuen Rings
  - fremde Nachricht
  - Join-Nachricht
  - Token-Loss-Timeout

Totem: Membership-Algorithmus (3)

- Lokal aufbewahrte Werte (u.a.)
  - *my_aru_count* / *my_last_aru*
  - *my_proc_set* / *my_fail_set*
  - *consensus* (Boolean-Array)
  - *my_trans_memb*

Totem: Membership-Algorithmus (4)

- Aufbau der Join-Nachricht

type	*sender_id*	*ring_seq*	*proc_set*	*fail_set*	*rotation_count*

  - Ziel: Konsens über *proc_set* und *fail_set*

- Aufbau der Configuration Change-Nachricht

ring_id	*seq*	*conf_id*	*memb*

  - nur lokal!

- Aufbau des Commit-Tokens

type	*ring_id*	*token_seq*	*seq*	*aru*	*aru_id*	*memb_list...*

Totem: Membership-Algorithmus (5)

- Operational State, Wechsel in den Gather State
  - Token-Loss-Timeout
  - fremde Nachricht (*my_proc_set* wird ergänzt)
  - Empfangsfehler (Erkennung durch *my_aru_count*)
  - Join-Nachricht (s.u.)
- Operational State, Ignorieren einer Join-Nachricht
  - Empfänger ist in *fail_set* enthalten
  - *ring_seq* ist kleiner als die Ring-Sequenznummer des Empfängers

Totem: Membership-Algorithmus (6)

- Gather State
  - Ziel: möglichst große Mitgliedergruppe
  - Join-Nachrichten übermitteln Informationen über funktionierende und fehlerhafte Prozessoren
  - beim Senden einer Join-Nachricht: Setzen von Join-Timeout und Konsens-Timeout

Totem: Membership-Algorithmus (7)

- Gather State, Empfang einer Join-Nachricht
  - Aktualisieren von *my_proc_set* / *my_fail_set*, eventuell Verwerfen von *consensus*
  - Aktualisieren von *consensus* (falls *proc_set* / *fail_set* gleich)
  - eigene ID in *fail_set* enthalten: Sender wird in *my_fail_set* eingetragen
- Gather State, Ignorieren einer Join-Nachricht
  - Sender kommt in *my_fail_set* vor
- Gather State, Join-Timeouts
  - bei Ablauf Neuübertragung

Totem: Membership-Algorithmus (8)

- Gather State, Konsens falls...
  - Join-Nachrichten mit passenden *proc_set* / *fail_set* von allen Prozessoren der Menge *my_proc_set* \ *my_fail_set*
  - Commit-Token mit passender Mitgliedschaft
- Gather State, Konsens-Timeout
  - alle Prozessoren aus *my_proc_set* ohne passende Join-Nachrichten werden *my_fail_set* hinzugefügt

Totem: Membership-Algorithmus (9)

- Gather State, nachdem Konsens erreicht
  - Setzen des Token-Loss-Timeout
  - Warten auf das Commit-Token
  - Representative erstellt Commit-Token
- Gather State, Empfang des Commit-Token
  - Verschiedene Prüfungen, eventuell Commit-Token verwerfen
  - ansonsten Eintragen von Informationen zum alten Ring
  - Wechsel in den Commit State

Totem: Membership-Algorithmus (10)

- Gather State, Token-Loss-Timeout
  - bei Ablauf: neuer Konsens
  - falls neuer Konsens identisch mit dem alten: der Prozessor, der das Commit-Token am seltenstem weitergereicht hat, wird zu *my_fail_set* hinzugefügt, neuer Konsens

Totem: Membership-Algorithmus (11)

- Commit State
  - zweiter Umlauf des Commit-Tokens
  - Bildung von *my_trans_memb*
  - Wechsel in den Recover State
- Commit State, mögliche Fehler
  - Join-Nachricht mit Ring-Sequenznummer größer der des neuen Rings: Wechsel in den Gather State
  - Token mit Ring-Sequenznummer kleiner der des neuen Rings wird verworfen

Totem: Membership-Algorithmus (12)

- Recover State
  - nach dem zweiten Umlauf wandelt der Representative das Commit-Token in ein reguläres Token um
  - Recovery-Algorithmus läuft ab, abschließend Wechsel in den Operational State

Totem: Recovery-Algorithmus

- Ziel: noch nicht ausgelieferte alte Nachrichten wiederherstellen
- Sechs Schritte
  - Nachrichtenaustausch mit Mitgliedern des alten Rings
  - Auslieferung in agreed oder safe order (falls möglich)
  - Wechsel in Übergangskonfiguration basierend auf *my_trans_memb*
  - Auslieferung von Nachrichten, die jetzt ausgeliefert werden können
  - Wechsel in die neue Konfiguration
  - Wechsel in den Operational State

SecureRing: Übersicht

- Prinzip wie bei Totem
- Schutz vor byzantinischen Fehlern
  - (unzuverlässiger) Fehlerdetektor
  - Voraussetzung: mindestens $\lceil (2n + 1) / 3 \rceil$ korrekte Prozessoren
- Digitale Signatur von Token, Commit-Token und Join-Nachrichten
- Token wird wie Nachrichten gemulticastet
- Token enthält vorheriges Token und eine Übersicht über die vom letzten Token-Besitzer gesendeten Nachrichten

SecureRing: Fehlerdetektor

- Ideale Eigenschaften
  - “Eventual Strong Byzantine Completeness”
  - “Eventual Strong Accuracy”
- Funktion
  - Überwachen von Nachrichten
  - Ausgabe: Liste von fehlerhaften Prozessoren

SecureRing: Fehlerdetektor (2)

- Vier Arten byzantinischer Fehler werden erkannt
  - mutierte Nachrichten
  - falsch aufgebaute Nachrichten
  - Unterlassen von Empfangsbestätigungen
  - Unterlassen des Sendens von benötigten Nachrichten

SecureRing: Fehlerdetektor (3)

- Mutierte Nachrichten
  - im Token gespeichertes vorheriges Token passt nicht zum wirklichen vorherigen Token
  - meherer Join-Nachrichten mit gleicher ID aber unterschiedlichem Inhalt von einem Prozessor

SecureRing: Fehlerdetektor (4)

- Falsch aufgebaute Nachrichten
  - Prüfung anhand der Felder
  - Token-Sequenznummer ⇔ Zahl gesendeter Nachrichten
  - *aru* und *rtr_list* müssen gegenseitig konsistent sein
  - *aru* darf nicht kleiner werden
  - möglicher Angriff: Erhöhung des seq-Felds und Vornehmen entsprechender Nachrichteneinträge, aber kein Senden von Nachrichten

SecureRing: Fehlerdetektor (5)

- Unterlassene Nachrichtenbestätigungen
  - *aru*-Feld bleibt länger unverändert oder dieselbe ID länger in *rtr_list*

SecureRing: Fehlerdetektor (6)

- Unterlassenes Senden: verschiedene Timeouts
  - Token-Loss-Timeout
  - möglicher Angriff: selektiver Multicast des Tokens, dagegen hilft Token Retransmission-Timeout (kürzer als Token-Loss-Timeout!)
  - Konsens-Timeout

Zusammenfassung

- Zwei Dienste zur Nachrichtenübermittlung in verteilten Systemen
  - eines (Totem) mit Erkennung nicht-byzantinischer Fehler durch Timeouts
  - das andere (SecureRing) mit Resistenz gegen bestimmte byzaninische Fehler durch Fehlerdetektor
- Aufrechterhaltung des Betriebs durch Ausschluss fehlerhafter Prozessoren