# 10 Lastverteilung

- Last tritt auf als
  - Rechenlast
  - Speicherlast
- ein Lastverbund ist eine Zusammenfassung mehrerer Rechner zur Aufnahme der aggregierten Last.
  - Verteilung auf Rechenserver oder
  - auf Dateiserver (siehe Kapitel 8)

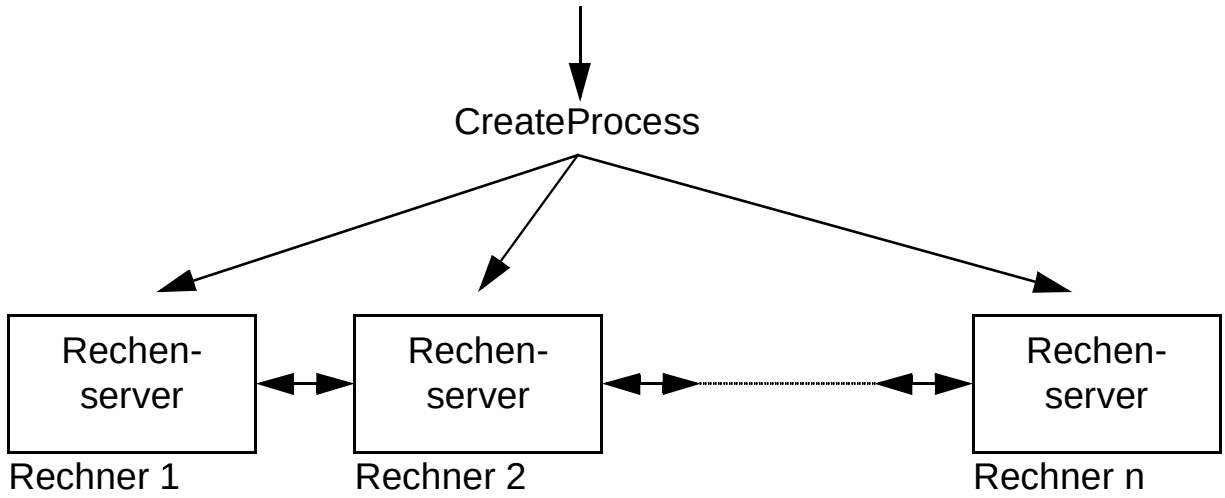


**Bild 10.1** Rechenlastverbund

- Fragestellungen:
  - wo wird ein neu zu erzeugender Prozess tatsächlich erzeugt?
  - soll ein bereits laufender Prozess zu einem anderen Prozessor wechseln und wenn ja, zu welchem?

# 10.1 Workstationmodell

- der Lastverbund basiert auf vernetzten privaten oder öffentlichen Workstations.
- jede Workstation kennt zwei Zustände:
  - belegt
  - idle

- Vorteile:
  - einfaches, leicht zu verstehendes Modell.
  - Benutzer verfügen über feste Ressourcen.
  - Benutzer verfügen über feste Antwortzeiten.
  - Benutzer haben hohe Autonomie.

- Nachteile:
  - Ressourcen sind beschränkt, auch wenn mehr gebraucht würden.
  - Ressourcen bleiben oft ungenutzt, weil der Benutzer gerade nichts oder wenig tut.

- Beobachtungen (an Universitäten) zeigen, dass selbst zu normalen Arbeitszeiten bis zu 30% der Workstations idle sind.
- Idee:
  - überlastete Workstations entlasten und „freie“ Workstations belasten.
  - die ungenutzte Rechenleistung durch geeignete Mechanismen zur Verfügung stellen.
  - Beispielsystem: Condor.

# Auffinden arbeitsloser Workstations

- Kriterien zur Erkennung arbeitsloser bzw. idle Workstations:
  - Benutzer ist an der Maschine angemeldet, genügt meist nicht.
    - selbst wenn kein Benutzer angemeldet ist, können viele Dämon-Prozesse laufen.
    - ist ein Benutzer angemeldet, muss er nicht unbedingt die Maschine belasten.
  - deshalb: eine Maschine ist arbeitslos, wenn:
    - keine benutzer-initiierten Prozesse laufen oder
    - Benutzerprozesse erzeugen keine Prozessorlast und es gab über einen gewissen Zeitraum keine Benutzerinteraktion.

- Finden arbeitsloser Workstations:
  - *Klientengesteuert*
    - ein Auftrag, mit den dazugehörigen Ressourcenanforderungen, wird an alle potentiellen Rechenserver geschickt.
    - die arbeitslosen Workstations melden sich zurück.
    - die Antwort kann entsprechend der jeweiligen lokalen Last verzögert sein.
    - der Klient erteilt einer der Stationen den Auftrag, wahrscheinlich der ersten, die sich meldet.
  - *Servergesteuert*
    - zentrale Variante
      - wird eine Workstation idle, meldet sie sich bei einer zentralen Registratur mit Namen, Adresse, Eigenschaften und Ressourcen.
    - dezentrale Variante
      - die Workstation schickt die Idle-Meldung an alle potentiellen Klienten.

## Transparente Ausführung des Rechenauftrags

- für den Benutzer, dessen Workstation Rechenlast auslagert, soll es transparent sein, wo die Last bearbeitet wird.
- es darf keinen Unterschied machen, ob ein Prozess lokal oder entfernt ausgeführt wird.
- der entfernte Prozess muss auf seinem Wirts-Rechenknoten die Umgebung der Heimat-Workstation vorfinden.
  - d.h. gleiche Sicht auf das Dateisystem,
  - gleiches Arbeitsverzeichnis,
  - gleiche Umgebungsvariablen,
  - gleichwertige Ressourcen.

- bei Bibliotheksfunktionen oder Systemaufrufen muss entschieden werden, ob diese Aufrufe vom Wirts- oder vom Heimatrechner bearbeitet werden:
  - *Lokale Ausführung*
    - alle Ein-/Ausgabeoperationen des Benutzers müssen auf der Heimat-Workstation ausgeführt werden
    - d.h. alle Aktionen bzgl. Tastatur, Maus, Bildschirm und Lautsprecher.
  - *Entfernte Ausführung*
    - alle Systemaufrufe, die sich auf CPU-Zugriffe oder auf Hauptspeicherzugriffe zurückführen lassen, müssen auf dem entfernten Rechenknoten ausgeführt werden.
    - temporäre Dateien sollten auf dem entfernten Knoten angelegt werden.
    - zeitbehaftete Aufrufe sind ohne systemweite globale Zeit oder zusätzliche Synchronisation kritisch.

- die Workstation ist nicht mehr idle:
  - der Benutzer kehrt zurück.
  - ein Ereignis stösst wartende Prozesse an.

  - was passiert mit einem ausgelagerten Prozess:
    - *Weiterrechnen*
      - für den Gast ändert sich, außer reduzierter Geschwindigkeit, nichts.
      - der Besitzer wird durch die zusätzliche Last beeinträchtigt.
      - er stellt deshalb seine Maschine evtl. nicht mehr für externe Prozesse zur Verfügung.
    - *Auftrag terminieren*
      - der Gastprozess wird abgebrochen.
      - die bis zu diesem Zeitpunkt verrichtete Arbeit geht verloren, falls Zwischenergebnisse nicht gesichert wurden.
    - *Zustand retten und terminieren*
      - der Gastprozess wird aufgefordert sich selbst herunterzufahren.
      - Ergebnisse und Zwischenzustände sind entsprechend zu sichern, bevor der Prozess terminiert.
    - *Migrieren*
      - der Prozess wird auf eine andere Maschine verlagert.
      - der Prozess muss seine gesamte Umgebung „mitnehmen“, z.B. offene Dateien, Kommunikationskanäle und prozessspezifische Betriebssystem-Ressourcen.

  - ein abgeschlossener Gastprozess sollte die Wirtsmaschine so hinterlassen, wie er sie bei seiner Ankunft vorgefunden hat.
    - d.h. temporäre Dateien löschen, Datenstrukturen aufräumen und Kommunikationskanäle schliessen, etc..

# 10.2 Prozessorpool-Modell

- ein Pool von Prozessoren steht zur Verfügung [Tan95].
- die Benutzer interagieren mit dem Pool über Grafikterminals.
- sie erhalten dynamisch eine jeweils passende Anzahl von Prozessoren.
- welche und wieviele Prozessoren ist zum Benutzer hin transparent und kann nicht beeinflußt werden.

- der Prozessorpool eignet sich besonders bei rechenintensiven Anwendungen.
- der Parallelitätsgrad muss jedoch „mitspielen“.
  (siehe Amdahl´s Law)
- beide Modelle können kombiniert werden:
  - rechenintensive Anwendungen auf dem Prozessorpool
  - Anwendungen mit viel Benutzerinteraktion auf Workstations

# 10.3 Verteilung von Rechenlast

- Problemstellungen:
  - auf welchem Prozessor oder welcher Workstation soll ein Prozess ausgeführt werden.
  - die Rechenlast soll möglichst gleichmäßig verteilt sein.

- Lastverteilungsalgorithmen:
  - statische Verfahren nutzen a priori bekannte Parameter.
  - dynamische Verfahren basieren auf dem gerade gemessenen Zustand des Gesamtsystems.

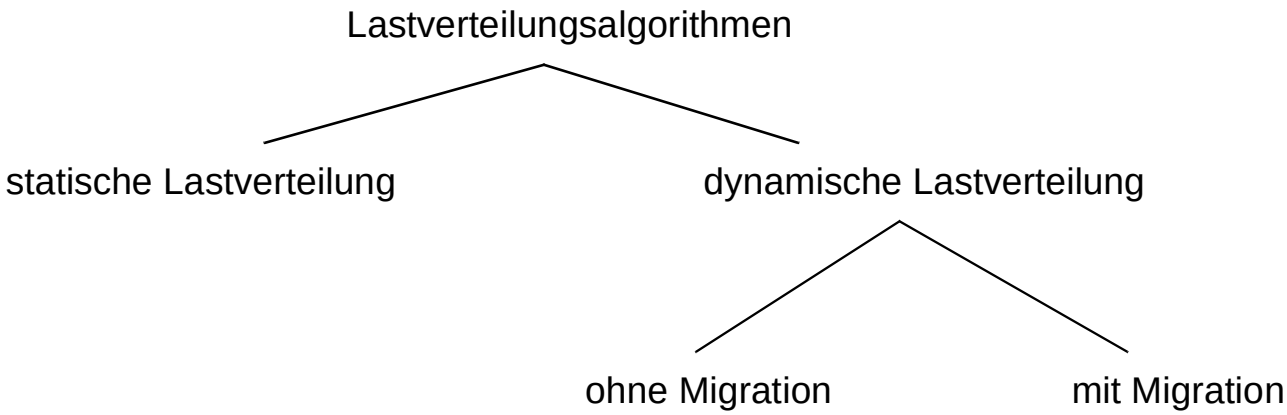


**Bild 10.2** Lastverteilungsverfahren

# 10.3.1Statische Lastverteilung

- meist werden deterministische Verfahren angewendet.
  - für ein Programm sind dessen Anforderungen an CPU, Speicher und Nachrichtenaufkommen bekannt.
  - anhand der Anforderungen wird jeder beteiligte Prozess einem bestimmten Prozessor zugeteilt.
- Es gibt sehr viele Algorithmen zur statischen Verteilung [Gos91], [Tan95].

**graphenbasierte Methode bzgl.**
**minimaler Kommunikationskosten**

- es wird ein gewichteter Graph erstellt, dessen Kantengewichte die Kommunikationskosten zwischen den Knoten repräsentieren.
- die Knotengewichte repräsentieren die Prozessorlast.
- Aufgabe:
  - finde eine möglichst günstige Partitionierung des Graphen, bei der die Kommunikationskosten minimiert sind und die Prozessoren nicht überlastet werden.

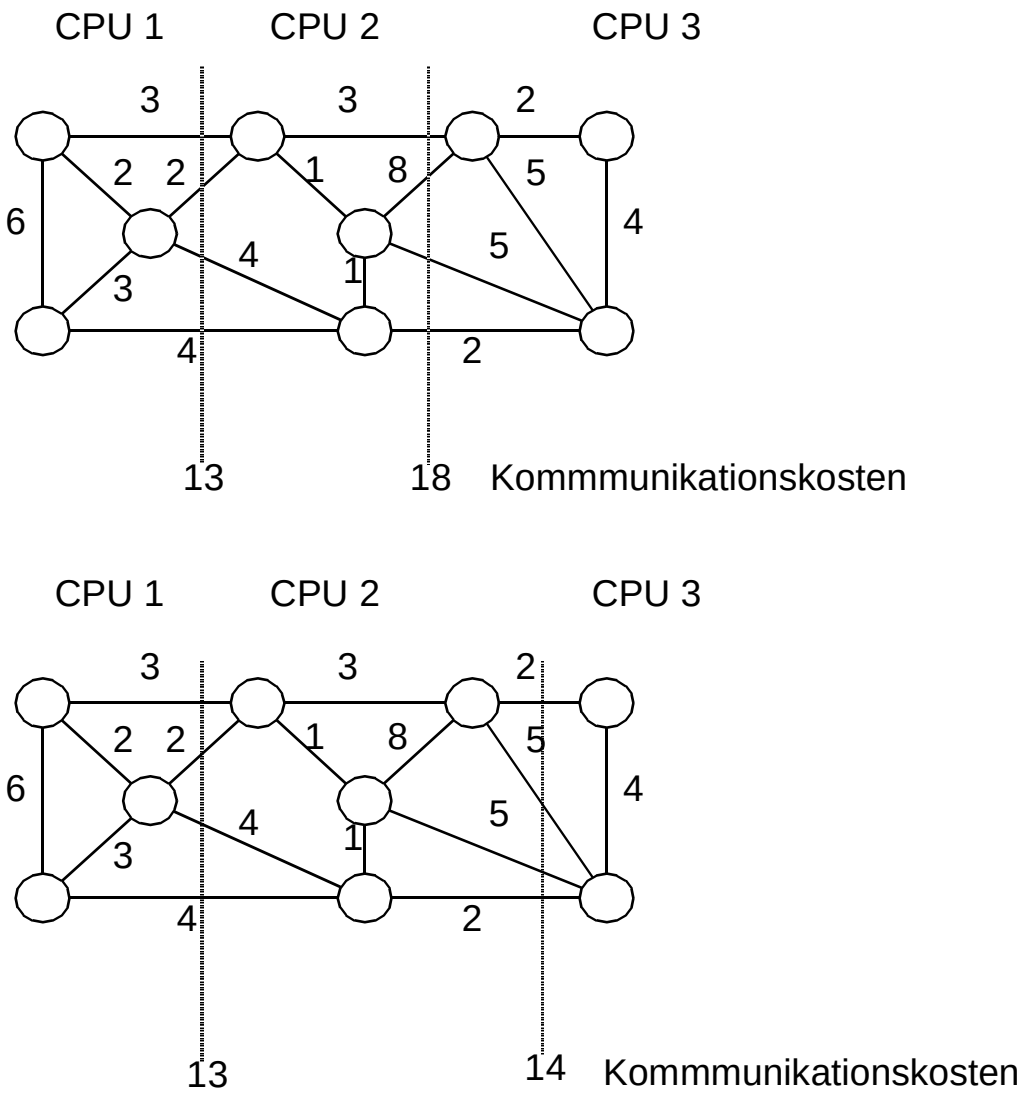


**Bild 10.3** Graphentheoretischer Verteilungsalgorithmus [Tan95]

- Problem der statischen Lastverteilung:
  - die Anforderungsprofile der Algorithmen müssen bekannt sein.
  - bei immer wiederkehrenden Durchläufen von Algorithmen ist dies möglich.
  - Beispiele:
    - Wettervorhersage
    - Simulationsverfahren

# 10.3.2Dynamische Lastverteilung

- die momentane Auslastung der Rechenknoten ist ausschlaggebend für die Zuteilung von Aufträgen.
- ein zusätzlich bekanntes Anforderungsprofil kann hilfreich sein.

- zentrale Komponente der dynamischen Lastverteilung ist die Lastmessung von:
  - Prozessorauslastung
  - Speicherauslastung
  - Kommunikationslast

- Metriken für Prozessorauslastung:
  - Gesamtanzahl der Prozesse.
  - Anzahl der Prozesse, die sich im Zustand `running` oder `ready` befinden.
  - Zeitanteil, den der Prozessor im letzten vergangenen Zeitintervall idle war.
  - mittlere Bedienzeit der Prozesse.
    - d.h. mittlere Verweildauer eines Prozesses im Zustand `ready`.

- Metriken für Speicherauslastung:
  - Anzahl der freien Speicherseiten im Hauptspeicher des Rechners.
  - Anzahl der freien Speicherseiten im virtuellen Speichersystem, d.h. auf den Festplatten des Rechners.

- Metriken für Kommunikationslast:
  - Anzahl gesendeter und empfangener Nachrichten pro Zeiteinheit.
  - Anzahl gesendeter und empfangener Bytes pro Zeiteinheit.

- die Werte müssen von einem Monitorprogramm während der Laufzeit für jeden Rechenserver erfasst und gespeichert werden.
- die Auswertung der erfassten Daten entspricht einem Optimierungsproblem.
  - Heuristiken sollen die Komplexität möglichst gering halten.
- die Monitorprogramme müssen einfach und wenig recheninten-siv sein, damit sie die Server nicht unnötig belasten und das Ergebnis verfälschen.

## Verteilung der Lastinformation

- *Klienten-initiiert*
  - ein Klient hat einen Rechenauftrag zu vergeben
  - er erfragt bei allen Knoten, bei einer Teilmenge der Knoten oder bei einem speziellen Knoten die Lastinformation ab.
  - er bewertet die erhaltene Information und wählt den Knoten mit der geringsten Belastung.
  - Dies heisst Lastabstossung.

- *Server-initiiert*
  - ein Server erfragt bei Unterbeschäftigung die Lastinforma-tion von Klienten ab.
  - die Bewertung nimmt der Server vor
  - er wählt den am meisten belasteten Knoten aus.
  - Dies heisst Lastanziehung.

- *Periodisch Server-initiiert*
  - alle Server schicken periodisch zu allen Knoten oder zu einer Teilmenge der Knoten ihre Lastinformation.
  - die Bewertung erfolgt lokal unabhängig bei jedem Empfänger oder
  - durch einen verteilten Bewertungsalgorithmus unter den Servern.
  - Hier kann Lastanziehung und Lastabstossung gemischt realisiert werden.

- bei allen Varianten kann die Information auch zentral bei einer Verwaltungsinstanz gehalten werden.

- Wichtiger Aspekt:
  - durch die zeitliche Verzögerung der Informationsverteilung und die Dauer der Bewertung kann die Lastinformation nie den aktuellsten Ist-Zustand wiedergeben.
  - die Bewertung kann zu falschen, überholten Schlüssen kommen.

## Architektur eines dynamischen Lastverteilungssystems

- Komponenten eines dynamischen Lastverteilungssystems:
  - Monitor zur Lastmessung
    - ermittelt die lokal anliegende Last.
  - Lastinformationskomponente
    - sammelt die gewonnenen Daten.
    - bereitet die Daten auf.
  - Transferkomponente
    - wählt die auszulagernden Prozesse aus.
    - sorgt dafür, dass die entfernte Umgebung angepasst, sowie Code und Daten verschickt werden.
  - Lokationskomponente
    - wählt den Wirtsrechner aus.
  - Verhandlungskomponente
    - bei verteilter Implementierung.
    - Austausch der Lastinformationen.
    - Verhandlung der Verteilung der Last untereinander.

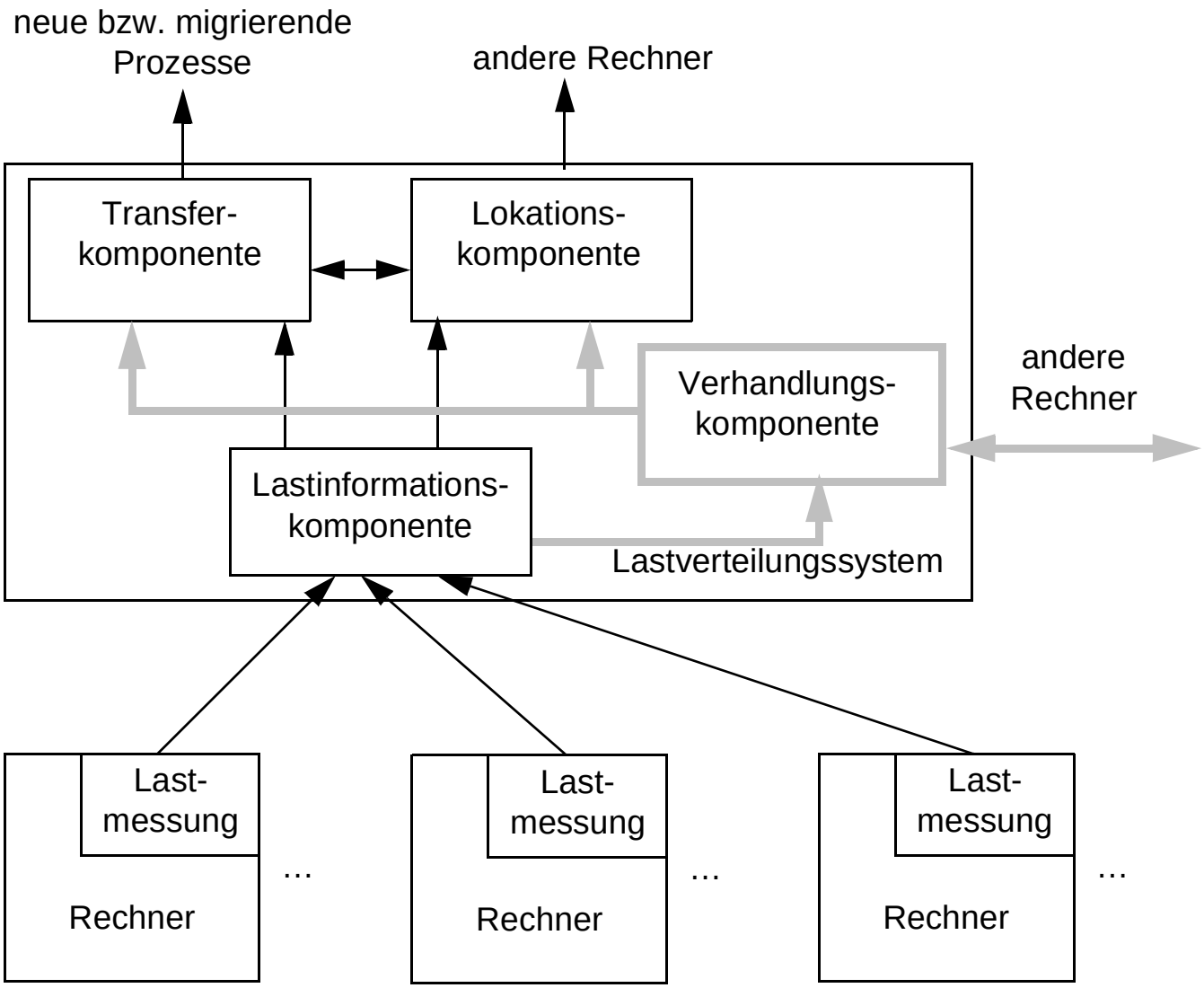


**Bild 10.4** Lastverteilungssystem [Gos91]

# 10.3.3Dynamische Lastverteilung ohne Migration

- Ziel:
  - neu entstehende Prozesse anhand der momentanen Lastbewertung auf dem „richtigen“ Rechenserver erzeugen.
  - Begriff: Initial Placement

**Klientenkontrollierte Lastverteilung**

- der Klient sucht selbst einen Rechenserver, auf dem er seinen Auftrag ausführen lassen will.
  - das Lastverteilungssystem, bis auf die Lastmessung, ist vollständig beim Klienten realisiert.
  - es gibt keine Verhandlungskomponente.

- Basisverfahren:

```
Broadcast an alle Rechenserver:
   Bitte Lastinformation abliefern;
Erwarte Rückantwort von allen Servern;
Ermittle optimalen Server;
CreateProcess (Code, Umgebung)
                  an Server senden;
Erwarte Antwort mit Prozeßidentifikation;
```

- Probleme:
  - Lastinformation ist kein stabiles Prädikat
    - die Bewertung muss das berücksichtigen.
  - die gleichzeitige Anfrage mehrerer Klienten provoziert Überlastung
    - alle Klienten erhalten dieselbe Lastinformation und wählen möglicherweise den gleichen Server.
  - schlecht skalierbar wegen möglicherweise großer Zahl Broadcasts.

- Gute praktische Ergebnisse werden mit einfachen Verfahren erzielt:
  - Verfahren 1:
    - CreateProcess wird an einen zufällig ausgewählten Server geschickt.
    - ist dieser Server überlastet, schickt er den Aufruf nach dem gleichen Prinzip weiter.
  - Verfahren 2:
    - der Wirtsrechners wird aus einer zufällig ausgewählten kleinen Menge von Repräsentanten gewählt.

## Serverkontrollierte Lastverteilung

- *Servergruppe mit Election*
  - ein Klient schickt einen CreateProcess-Auftrag an alle Server in einer Gruppe.
  - die Server starten einen Election-Algorithmus, der die Last der Server als Auswahlkriterium verwendet.
  - der Server mit der geringsten Last erzeugt den Prozess
  - er schickt an den Klienten eine Bestätigung mit der Prozessidentifikation.

  - dieses Verfahren arbeitet bei kleinen Gruppen von Servern gut.
  - bei größeren Servergruppen wird die Election zum kritischen Punkt.

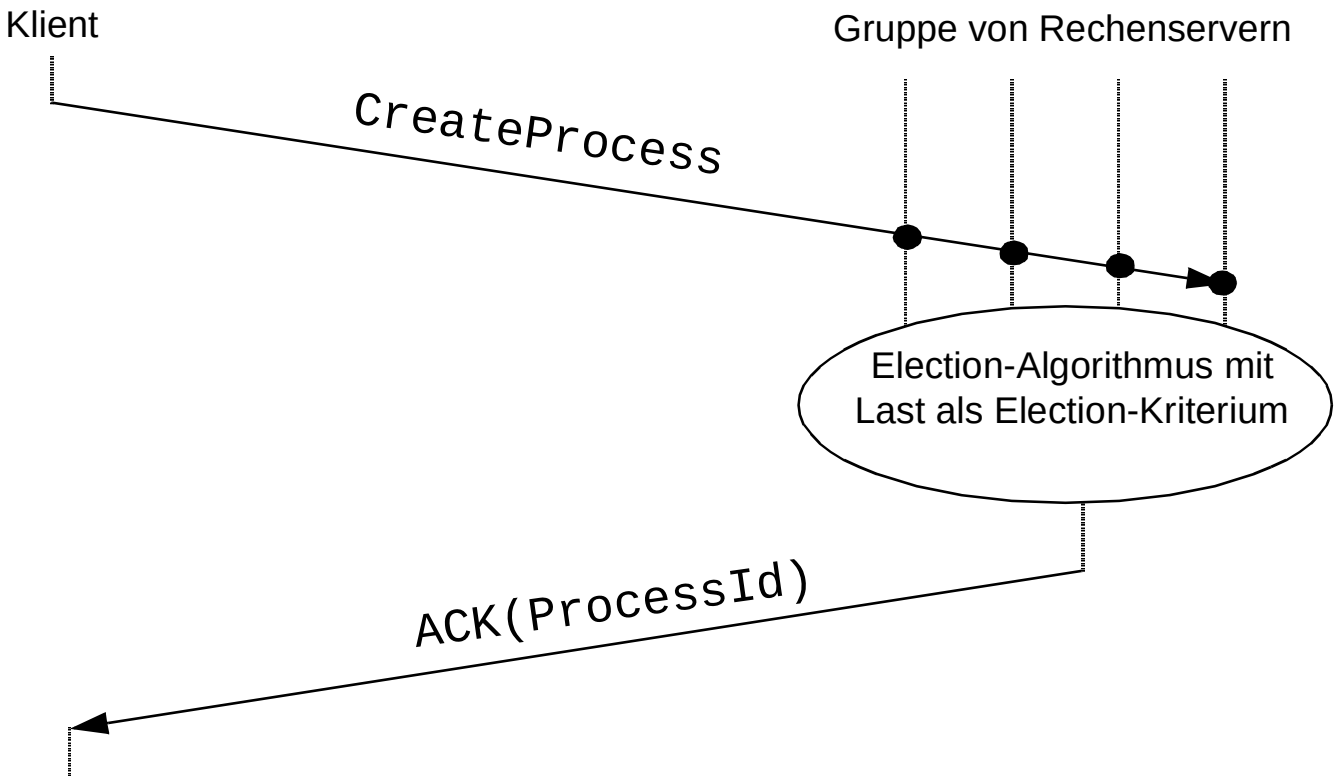


**Bild 10.5** Serverkontrolliertes Initial Placement mit Election

- Außer der Transferkomponente sind alle Komponenten auf den Servern implementiert.
- die Lokations- und die Verhandlungskomponente sind durch den Election-Algorithmus realisiert.

- *Delegation an einen Agenten*
  - der Klient stellt eine Anfrage an einen lokalen Agenten.
  - der Agent stellt nun eine Lastanfrage an einen Teil der Rechenserver.
  - der Agent spricht gezielt bestimmte Server an,
    - da er durch vorherige Anfragen bereits ein Bild über die Lastverteilung besitzt.
  - aus den erhaltenen Lastinformationen wählt der Agent einen Rechenserver aus, der den Auftrag erhält.

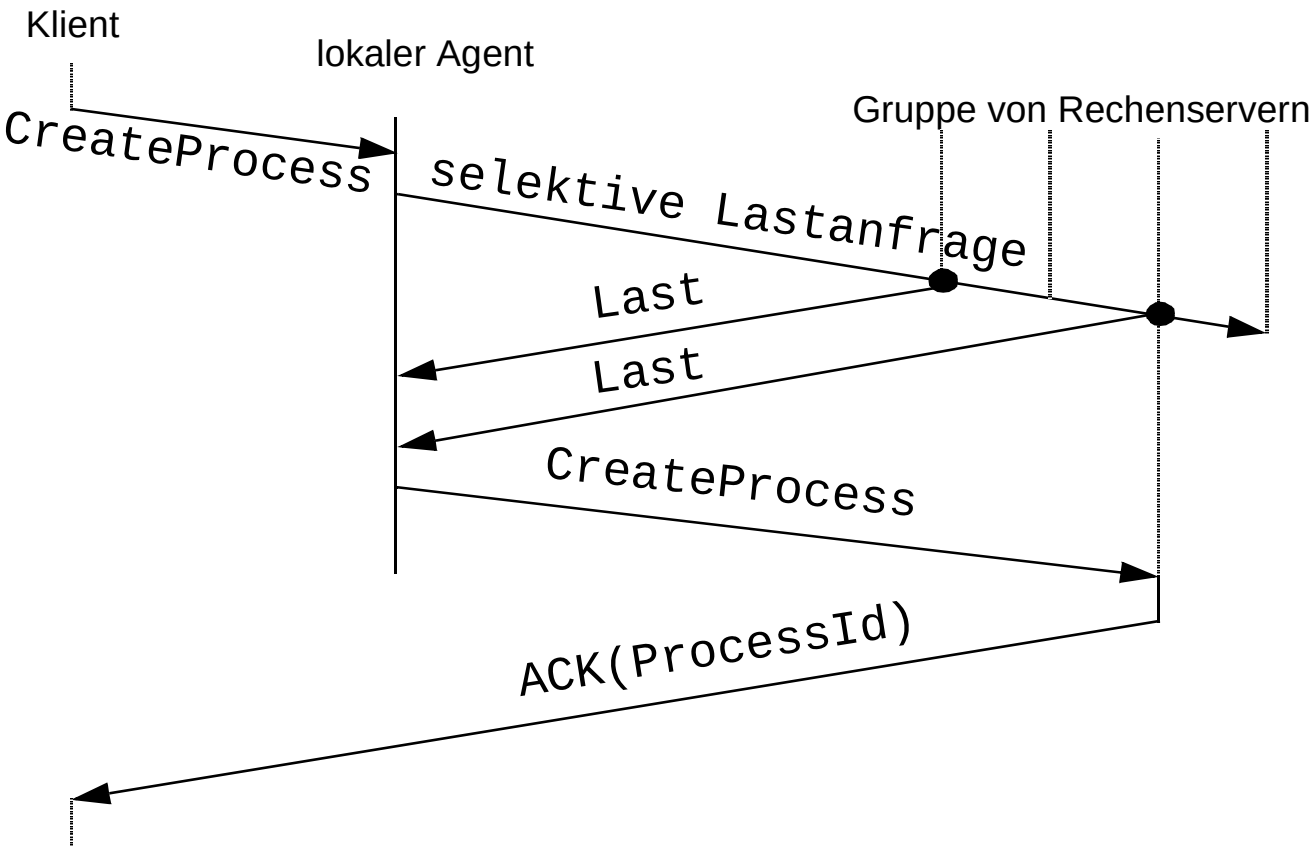


**Bild 10.6** Server-kontrolliertes Initial Placement mit lokalem Rechenserver

- durch die selektive Anfrage skaliert dieses Verfahren gut.

- der Agent realisiert die Lastinformations-, Lokations- und Transferkomponente.
- die Server verfügen über einen Monitor.
- falls mehrere Agenten untereinander Informationen austauschen, verfügen sie zusätzlich über Verhandlungskomponenten.

- In allen Szenarien können Klienten auch als Server agieren, so dass die Verfahren dann symmetrisch angelegt sein müssen.

# 10.3.4Dynamische Lastverteilung mit Migration

- Ziel:
  - Last dynamisch während der Laufzeit ausgleichen.
  - laufende Prozesse von einem stark belasteten Knoten auf einen schwach belasteten verlagern.

- Prinzipien:
  - *Lastabstoßung*
    - bei einem Quellknoten wird ein Lastschwellwert überschritten
    - der Knoten sucht einen anderen, weniger belasteten Knoten und
    - migriert Last zu diesem Zielknoten.

    - bei Überlast im Gesamtsystems:
      - Lastabstoßung führt zum Thrashing.
      - d.h. alle Knoten sind nur noch damit beschäftigt, Last abzustossen.

  - *Lastanziehung*
    - ein Knoten bietet anderen Knoten seine Ressourcen an, sobald sein Lastschwellwert unterschritten wird.
    - stark belastete Knoten können Prozesse zum unterbelasteten Knoten migrieren.
    - der sollte, um Thrashing zu verhindern, dem zustimmen.

- dynamische Lastverteilung mit Migration ist meist symmetrisch angelegt,
  - d.h. alle Rechner sind sowohl Klient, als auch Rechenserver.
  - alle Komponenten sind auf allen Knoten vorhanden.

## Ablauf einer Migration

- bei Migration laufender Prozesse muss extern exakt die aktuelle Sicht geschaffen werden.
  - der aktuelle Adressraum des Prozesses und sein interner Zustand sind zu erfassen und zu übertragen.
  - alle Kommunikationsbeziehungen des migrierenden Prozesses müssen intakt bleiben.

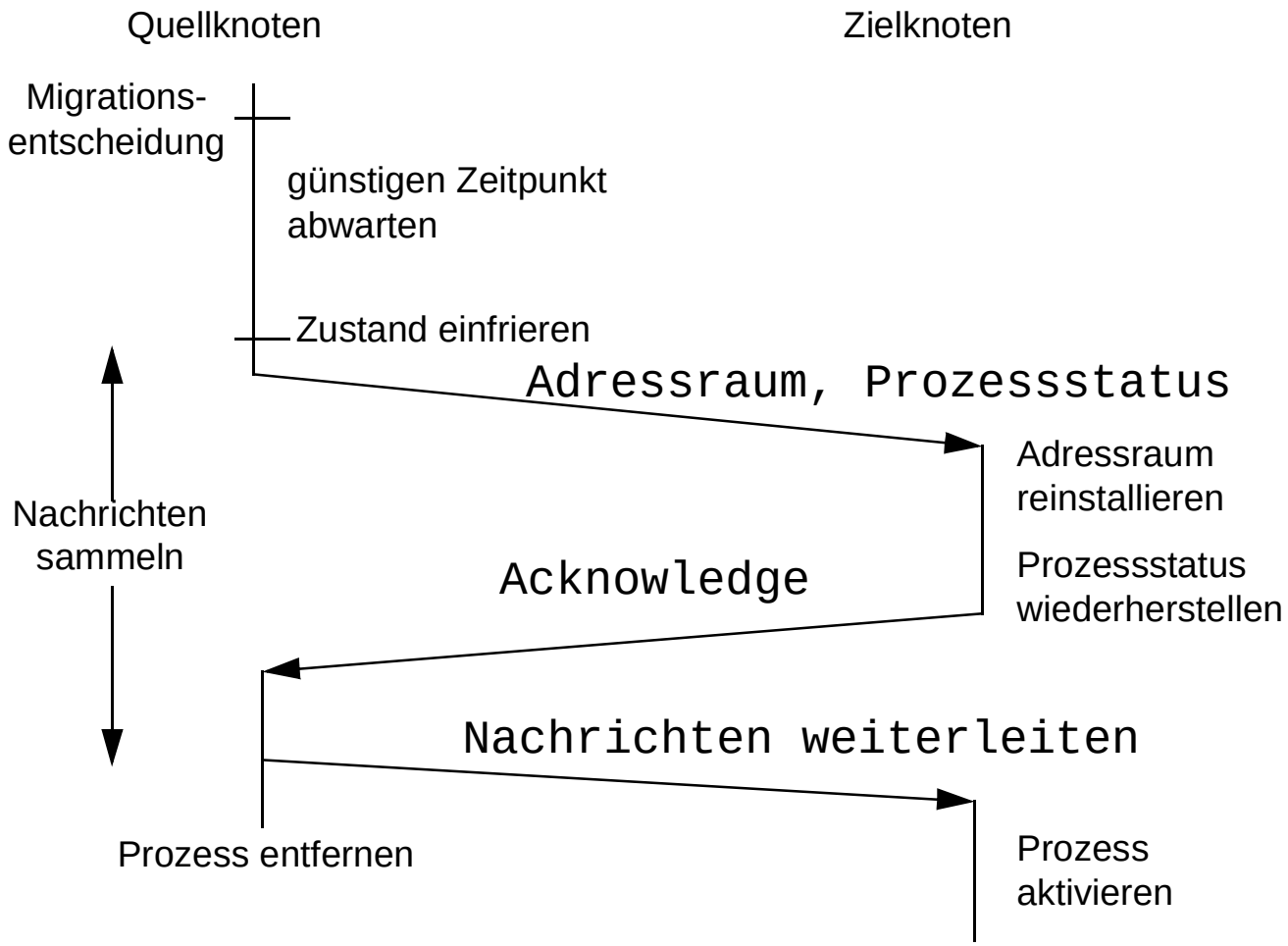


**Bild 10.7** Ablauf einer Prozessmigration

- während der Migration muss der Quellknoten Nachrichten für den migrierenden Prozess aufbewahren und sie ihm nach der Migration zuspielen.
- die Kommunikationskanäle müssen nach der Migration entweder auf den neuen Knoten zeigen, oder
- ein Durchreiche-Prozess (Proxy) verbleibt auf dem Quellknoten.

## effizientere Varianten

- *Pre-Copying*
  - die Übermittlung des Adressraums beginnt bereits zwischen der Migrationsentscheidung und dem Einfrieren des Prozesses.
  - nach dem Einfrieren müssen die während der Transferzeit neu bearbeiteten Seiten aktualisiert werden.

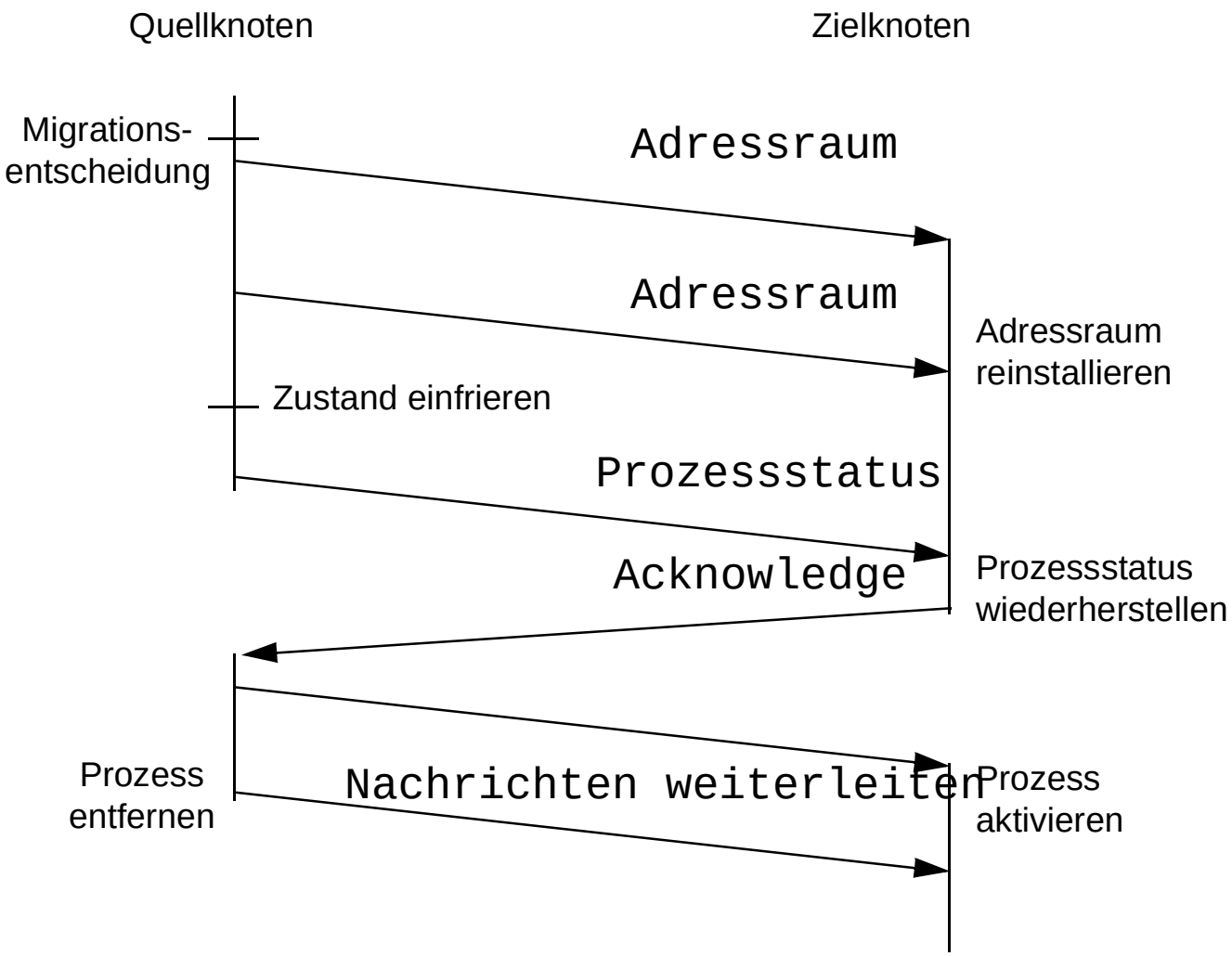


**Bild 10.8** Prozessmigration mit Pre-Copying

- Copy-on-Reference bzw. Post Copying
  - nach der Migrationsentscheidung werden alle Seitenzugriffe des Prozesses gesperrt.
  - der ersten Zugriff auf eine nicht vorhandene Seite löst einen Seitenfehler aus.
  - die benötigte Seite wird übermittelt.
  - es entsteht ein minimaler Arbeitsadressraum.
  - dieser wird zusammen mit dem Prozesszustand übertragen.
  - der Prozess wird auf dem Zielknoten gestartet.
  - Zugriffe auf noch nicht übertragene Seiten führen zu Seitenfehlern; die Seiten werden nachgeliefert.

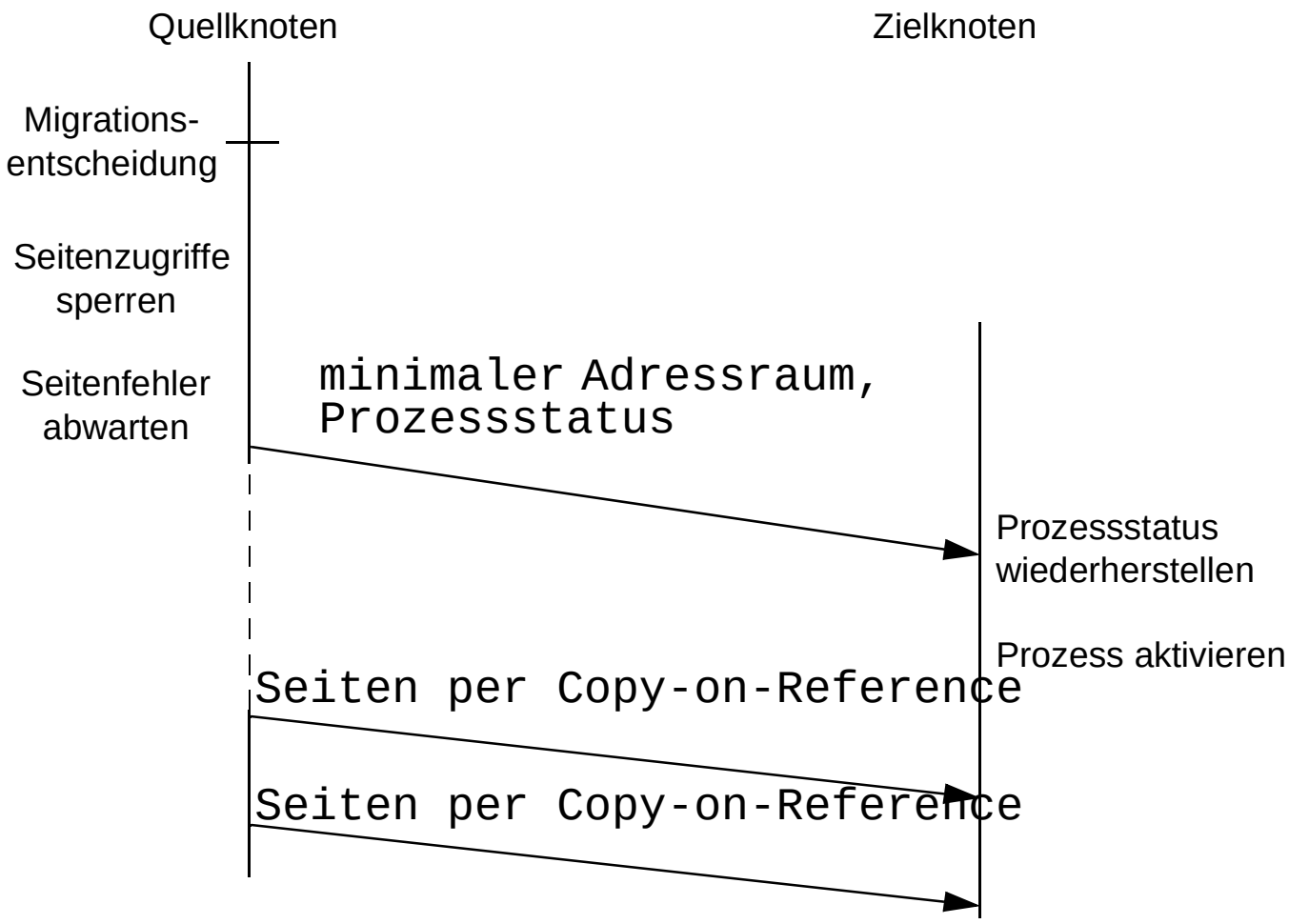


**Bild 10.9** Prozessmigration mit Copy-on-Reference