Recovery-oriented computing (ROC)

Datenbanken und Informationssysteme

Technische Universität Kaiserslautern
Lehrgebiet Datenverwaltungssysteme
Wintersemester 2005/2006

Benjamin Mock

Motivation

- "If a problem has no solution, it may not be a problem, but a fact - not to be solved, but to be coped with over time.“

  (Shimon Peres, israelischer Politiker)

→ Fehler in Computersystemen sind unvermeidbar

→ Fehler tolerieren oder kompensieren

Übersicht

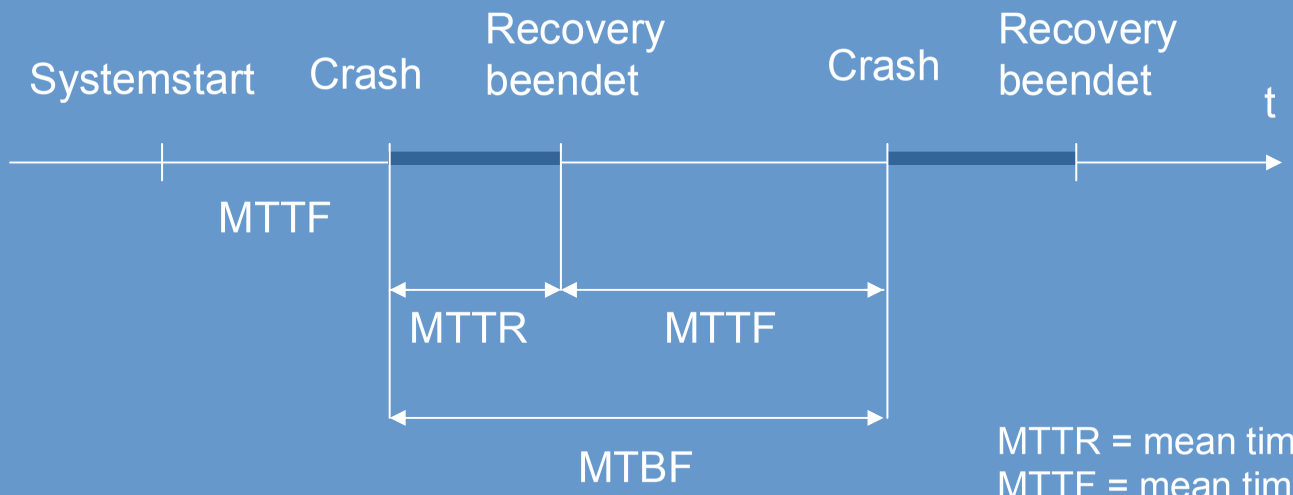
- Ziele
- Definition der Verfügbarkeit
- ROC in DBMS
- Fehlerklassen
- ROC-Techniken
- ROC-Bausteine
- Kosten/Nutzen
- Fazit

Ziele

- Ziele von ROC:
  - Erhöhung der Verfügbarkeit
  - Verringerung der total costs of ownership
    (Anschaffungskosten + Kosten für laufenden Betrieb)
  - Verbesserung der Administrierbarkeit
  - Trainingssystem

Verfügbarkeit

$$Verfügbarkeit = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$



MTTR = mean time to repair
MTTF = mean time to failure
MTBF = mean time between failures

ROC in Datenbankmanagementsystemen

- ACID-Eigenschaften werden gefordert
- Im Fehlerfall: jüngster transaktionskonsistenter Zustand wiederherstellen
- Sammlung redundanter Daten im Normalbetrieb
  - Protokollierung von Aktionen (Logging)
  - Anlegen von Sicherungspunkten

Fehlerklassen

- Transaktionsfehler
  - Fehler in Anwendung, Deadlock
- Systemfehler
  - Fehler in Systemsoftware, Stromausfall, menschliche Fehlbedienung
- Plattenfehler
  - Datenverlust im Festspeicher
- Katastrophen

  - Erdbeben, Überschwemmungen, Feuer

ROC-Techniken

- Modularisierung
  - Java Virtual Machine simuliert komplettes System
  - unabhängige und eigenständige Komponenten
- Redundanz
  - Kein Single point of failure
  - zusätzliche Festplatten, Lüfter, Netzwerkkarten …
  - Cluster-Systeme

ROC-Bausteine

- Rekursive Neustarts
  - vor allem für Java-Umgebung interessant
  - Tatsächliche Anwendung in Praxis: Mercury-Satelliten-Projekt
- Undo auf Systemebene
  - Wiederherstellen von Sicherungspunkten
  - Besonderheit: Replay-Schritt
- ROC auf Hardwarebene: ROC-1
  - Bisher nur Prototyp
  - Völlig anderer Ansatz als andere Bausteine

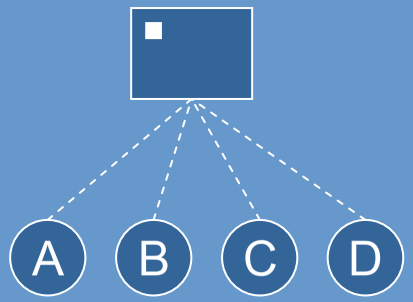
Rekursive Neustarts

- Viele Fehler lassen sich durch Neustart beheben
  - Systeme nicht auf solches Vorgehen ausgelegt
- Idee: Neustarts auf verschiedenen Ebenen (Recursive Restartability)
- Granularität: System, Teilsystem, Komponenten
- Zwei Ansätze:
  - Neustarts fehlerhafter Komponenten zur Fehlerbehebung
  - periodische Neustarts fehlerfrei laufender Komponenten zur Verjüngung

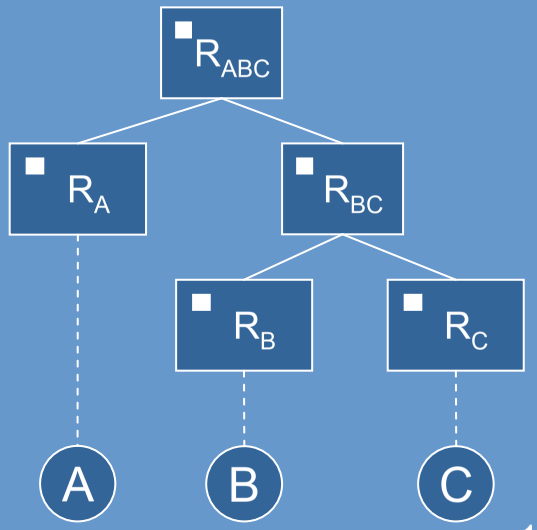
Neustart-Bäume

- Konventionelles System
  - Komponenten können nur gemeinsam neu gestartet werden

$MTTR_{system} \geq \max(MTTR_{komponenten})$



- System mit Recursive Restartability
  - Komponenten können einzeln und in Gruppen gestartet werden



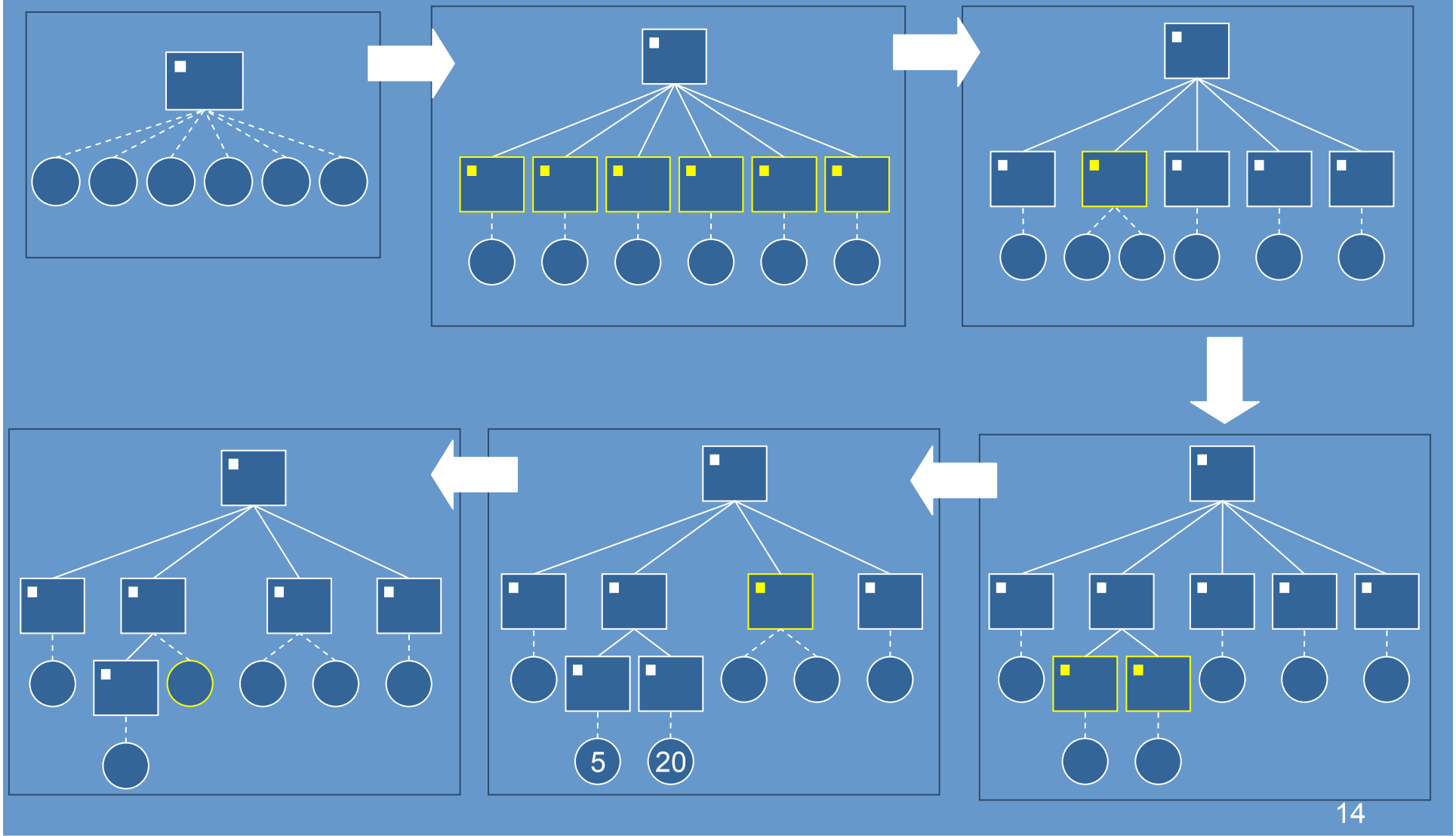
Beispiel für RR: Mercury-Bodenstation

- Kommunikation mit Satelliten
  - Datenübertragung nur in bestimmtem Winkel möglich (vier mal täglich für 15min)
- Aufbau aus kostengünstigen „commercial off-the-shelf Components“
  - Jede Komponente läuft in eigener Java Virtual Machine
  - Fehlerbeseitigung durch Administratoren durch einfaches Neustarten des Systems

Beispiel für RR: Mercury-Bodenstation II

- Fehlererkennung durch Pings („are you alive?“)
  - Einfach zu realisieren
  - Fehlererkennung wird automatisiert und benötigt keinen Administrator mehr
- Komponenten
  - lose gekoppelt
    - Keine Änderungen am eigentlichen Code nötig
  - Feine Granularität ⟺ viele Interfaces
  - Kosten für Neustrukturierung ⟺ Kosten für Ausfälle

Entwicklung eines Neustart-Baums



Verbesserung durch rekursive Neustarts

- Bis zu sechsfache Verbesserung der MTTR bei einzelnen Komponenten
- Etwa vierfache Verbesserung der MTTR für das Gesamtsystem

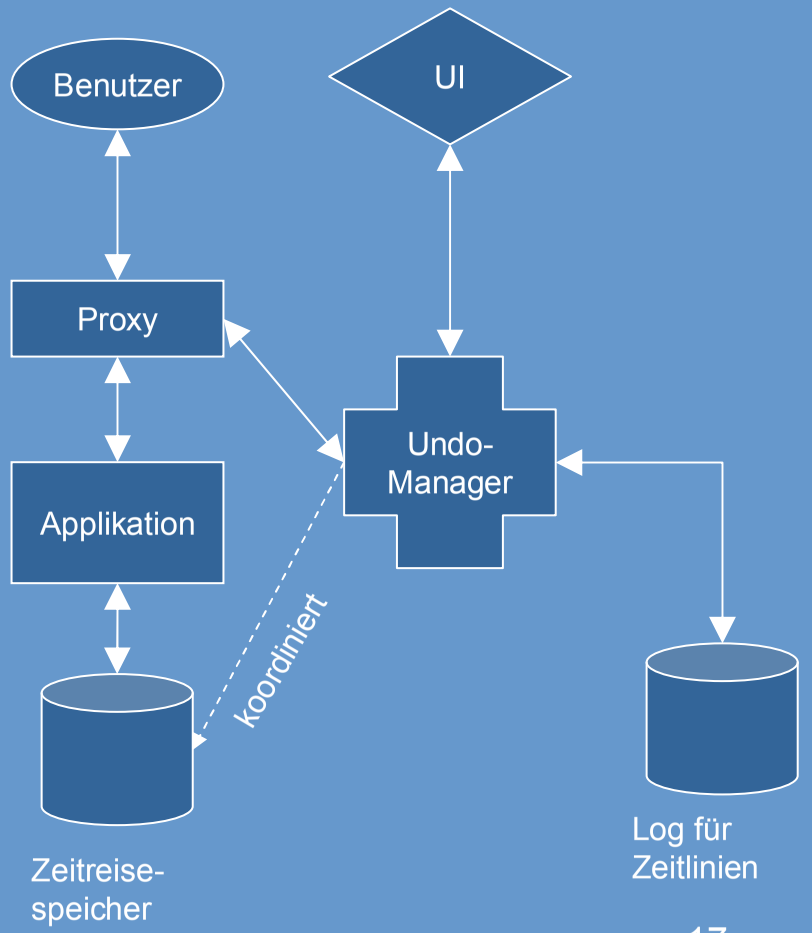
Komponenten:	msgbone	fedr	pbcom	ise/istr	istu
MTTF	1 Monat	10min	1 Monat	5h	5h
MTTR vorher in s	28,9	28,9	28,9	28,9	28,9
MTTR nachher in s	4,7	5	21,9	6,1	5,8

Undo auf Systemebene

- Administrierungsfehler nicht ausschließbar
- Automatisierungsironie
  - Automatische Behebung leichter Fehler
  - Trainingseffekt für Administratoren geht verloren
  - Schlechterer Umgang mit schweren Fehlern
- Unterstützt Versuchs-und-Irrtums-Methode

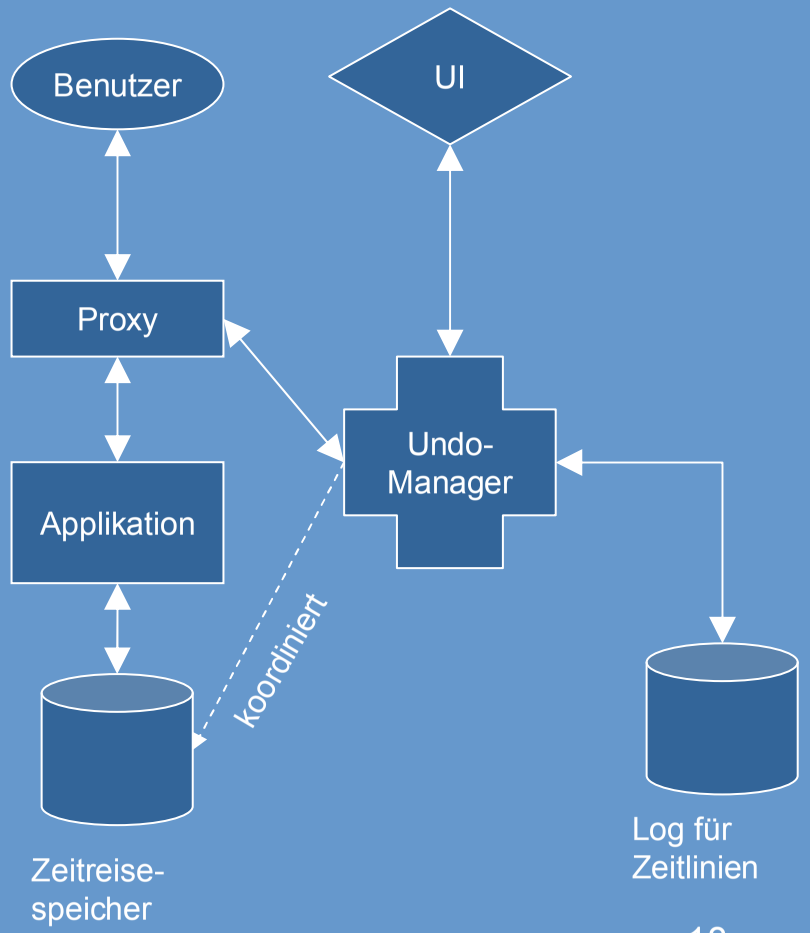
Undo auf Systemebene: Schema

- Proxy: hört Benutzeroperationen mit
- Zeitreisespeicher: Zustandsschnappschüsse
- Log: Speicherung der Intention der Benutzeraktion
- Undo-Manager: Herzstück und Koordinator



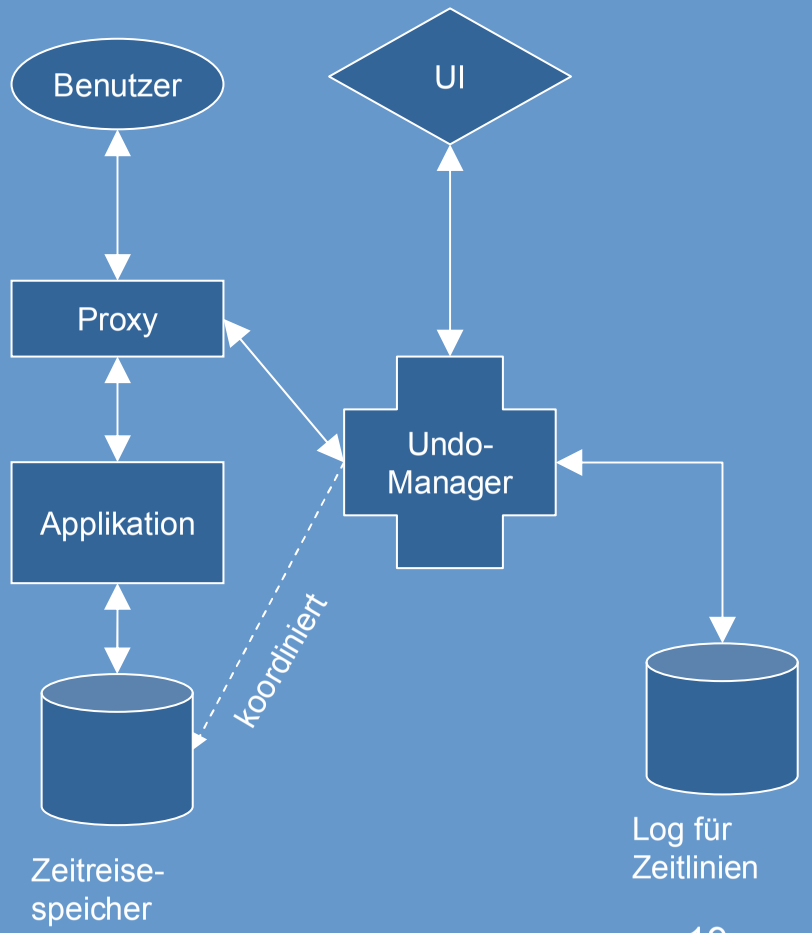
Rewind-Schritt

- Drei Schritte: 3 R‘s:
  - **Rewind**
  - Repair
  - Replay
- Rewind-Schritt
  - ähnlich Rollback in DBMS
  - Wiederherstellen eines Zustands aus Zeitreisespeicher



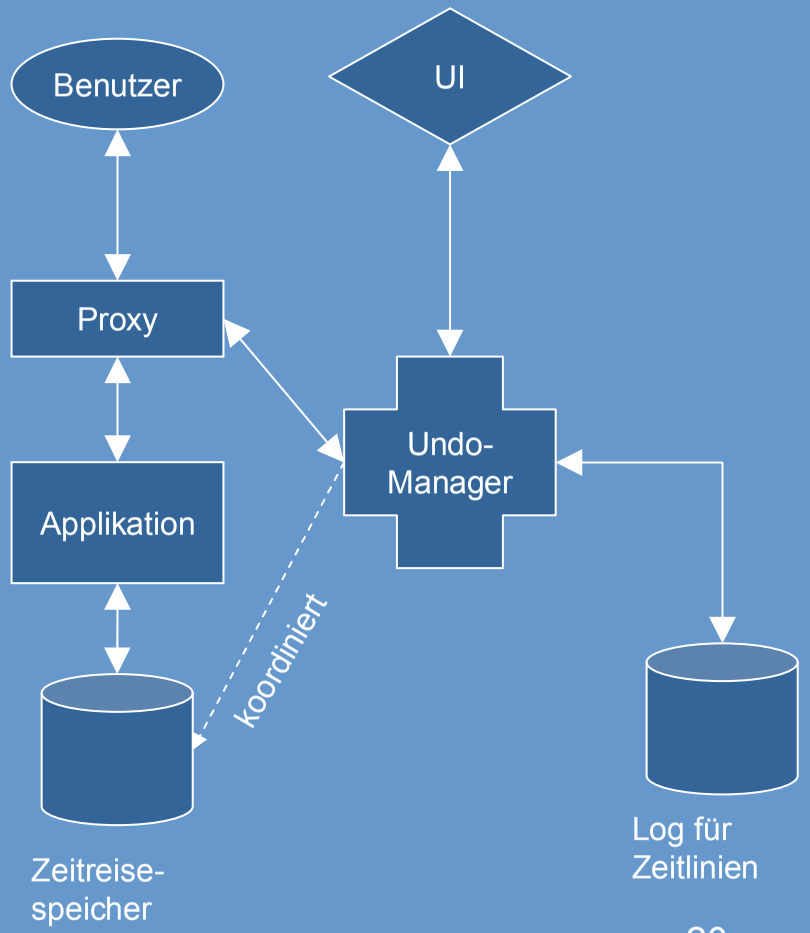
Repair-Schritt

- Drei Schritte: 3 R‘s:
  - Rewind
  - **Repair**
  - Replay
- Repair-Schritt
  - Änderungen oder Reparaturen vornehmen
  - Aktion unterlassen



Replay-Schritt

- Drei Schritte: 3 R‘s:
  - Rewind
  - Repair
  - **Replay**
- Replay-Schritt
  - Wiederholen aller Benutzeraktionen



Beispiel für Undo auf Systemebene

Achtung: „*etwas*“ unwissenschaftlich!

Marty McFly

Doc Brown

DeLorean

Rewind

- Aufbruch zur Zeitreise (**Rewind-Schritt**)

Repair

- Veränderung in der Vergangenheit (**Repair-Schritt**)

Replay

- Rückkehr in die Gegenwart (**Replay-Schritt**)

Undo auf Systemebene: Probleme

- Probleme:
  - Externe Konsistenz
  - Kein Logging der Intention der Reparaturen möglich, deshalb auch kein Wiederherstellen für diese
  - Einschränkung der Benutzeraktionen auf festgelegtes Protokoll
  - Regelwerk für Vorrang verschiedener Undos (Administrator, Benutzer …)

ROC-1

- 64 Knoten
  - 266MHz-Pentium-II-Mobile-Prozessor
  - 18GB-SCSI-Festplatte
  - 256MB fehlerkorrigierendes DRAM
  - vier redundante 100Mb/s-Netzwerkkarten
  - 18MHz-Motorola-Diagnoseprozessor
- 16 First-Level-Switches
- zwei Gigabit-Switches

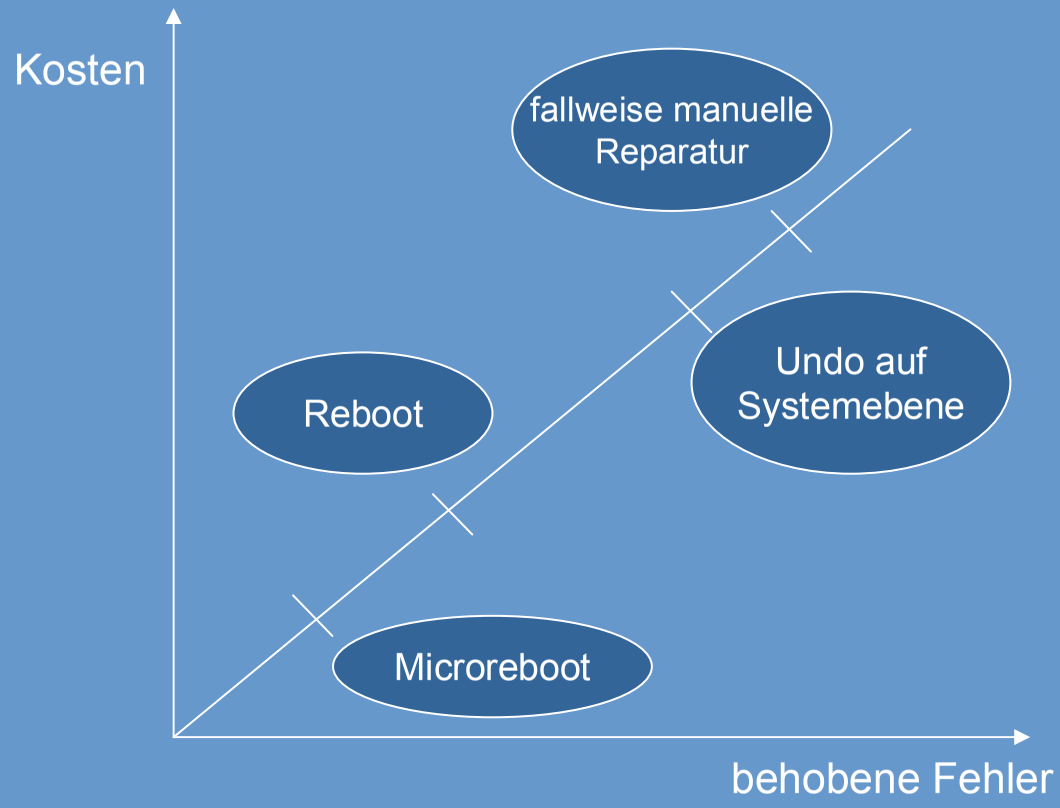
- Gesamtspeicherkapazität : 1,2TB
- passt in drei Racks

ROC-1 II

- Unterschiede zu „normalen“ Clustern
  - Verhältnis von Festplatte zu CPU: 1/1
  - billigere und stromsparendere Prozessoren
  - Zusammenschluss mit einem Diagnosesystem
- ROC-Techniken
  - physische Isolation
  - Redundanz
  - Selbsttest und Verifikation
  - Verbesserung der menschlichen Interaktion mit dem System

Kosten/Nutzen

- Keine perfekte Lösung möglich!



Fazit

- Bisherige Entwicklung und Forschung hauptsächlich zur Verbesserung der Verfügbarkeit durch Erhöhung der MTTF
  - Wirksameres Marketing?
  - Jetzt mehr Potenzial für ROC?
- Fehlerhafte Software zu entwickeln ist okay?
- Nur Kombination verschiedener ROC-Bausteine in Verbindung mit guter Software bringt wirkliche Verbesserung der Verfügbarkeit