Pseudocode für Crash−Recovery mit asynchronen Sicherungspunkten:

```
/* Basic data structures of page−model redo−winners recovery algorithms: */

type Page: record of
      PageNo: identifier;
      PageSeqNo: identifier;
      Status: (clean, dirty);
      Contents: array [PageSize] of char;
   end;
persistent var StableDatabase:
      set of Page indexed by PageNo;
var DatabaseCache:
      set of Page indexed by PageNo;
type LogEntry: record of
      LogSeqNo: identifier;
      TransId: identifier;
      PageNo: identifier;
      ActionType: (write, full−write, begin, commit, rollback, checkpoint);
      ActiveTrans: set of TransInfo;
         /* present only in log entries of type checkpoint */
      DirtyPages: set of DirtyPageInfo;
         /* present only in log entries of type checkpoint */
      UndoInfo: array of char;
      RedoInfo: array of char;
      PreviousSeqNo: identifier;
   end;
persistent var StableLog:
      ordered set of LogEntry indexed by LogSeqNo;
var LogBuffer:
      ordered set of LogEntry indexed by LogSeqNo;
persistent var MasterRecord: record of
      StartPointer: identifier;
      LastCP: identifier;
   end;
type TransInfo: record of
      TransId: identifier;
      LastSeqNo: identifier;
   end;
var ActiveTrans:
      set of TransInfo indexed by TransId;
type DirtyPageInfo: record of
      PageNo: identifier;
      RedoSeqNo: identifier;
   end;
var DirtyPages:
      set of DirtyPageInfo indexed by PageNo;
```



```
/* Actions during normal operation */

write or full−write (pageno, transid, seqno):
  DatabaseCache[pageno].Contents := modified contents;
  DatabaseCache[pageno].PageSeqNo := seqno;
  DatabaseCache[pageno].Status := dirty;
  newlogentry.LogSeqNo := seqno;
  newlogentry.ActionType := write or full−write;
  newlogentry.TransId := transid;
  newlogentry.PageNo := pageno;
  newlogentry.UndoInfo := information to undo update
     (before−image for full−write);
  newlogentry.RedoInfo := information to redo update
     (after−image for full−write);
  newlogentry.PreviousSeqNo :=
      ActiveTrans[transid].LastSeqNo;
  ActiveTrans[transid].LastSeqNo := seqno;
  LogBuffer += newlogentry;
  if pageno not in DirtyPages then
       DirtyPages += pageno;
       DirtyPages[pageno].RedoSeqNo := seqno;
  end /*if*/;

fetch (pageno):
  DatabaseCache += pageno;
  DatabaseCache[pageno].Contents :=
     StableDatabase[pageno].Contents;
  DatabaseCache[pageno].PageSeqNo :=
     StableDatabase[pageno].PageSeqNo;
  DatabaseCache[pageno].Status := clean;

flush (pageno):
  if there is logentry in LogBuffer with logentry.PageNo = pageno
  then
     force ( );
  end /*if*/;
  StableDatabase[pageno].Contents :=
     DatabaseCache[pageno].Contents;
  StableDatabase[pageno].PageSeqNo :=
     DatabaseCache[pageno].PageSeqNo;
  DatabaseCache[pageno].Status := clean;
  DirtyPages −= pageno;

force ( ):
    StableLog += LogBuffer;
    LogBuffer := empty;
```

```
begin (transid, seqno):
  ActiveTrans += transid;
  ActiveTrans[transid].LastSeqNo := seqno;
  newlogentry.LogSeqNo := seqno;
  newlogentry.ActionType := begin;
  newlogentry.TransId := transid;
  newlogentry.PreviousSeqNo := nil;
  LogBuffer += newlogentry;

commit (transid, seqno):
  newlogentry.LogSeqNo := seqno;
  newlogentry.ActionType := commit;
  newlogentry.TransId := transid;
  newlogentry.PreviousSeqNo :=
      ActiveTrans[transid].LastSeqNo;
  LogBuffer += newlogentry;
  ActiveTrans −= transid;
  force ( );
```


/* **Checkpointing and log truncation during normal operation** */

```
log truncation ( ):
  OldestUndoLSN :=
      min {i | StableLog[i].TransId is in ActiveTrans};
  SystemRedoLSN := min {DirtyPages[p].RedoSeqNo};
  OldestRedoPage := page p such that
      DirtyPages[p].RedoSeqNo = SystemRedoLSN;
  NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};
  OldStartPointer := MasterRecord.StartPointer;
  while OldStartPointer − NewStartPointer
          is not sufficiently large
  and SystemRedoLSN < OldestUndoLSN
  do
    flush (OldestRedoPage);
    SystemRedoLSN := min{DirtyPages[p].RedoSeqNo};
    OldestRedoPage := page p such that
        DirtyPages[p].RedoSeqNo = SystemRedoLSN;
    NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};
  end /*while*/;
  MasterRecord.StartPointer := NewStartPointer;

checkpoint ( ):
  log truncation ( );
  logentry.ActionType := checkpoint;
  logentry.ActiveTrans :=
      ActiveTrans (as maintained in memory);
  logentry.DirtyPages := DirtyPages;
  logentry.LogSeqNo := new sequence number;
  LogBuffer += logentry;
  force ( );
  MasterRecord.LastCP := logentry.LogSeqNo;
```

```
/* Simple three−pass crash−crash recovery */

restart ( ):
  analysis pass ( ) returns losers;
  redo pass ( );
  undo pass ( );

analysis pass ( ) returns losers, DirtyPages:
  cp := MasterRecord.LastCP;
  losers := StableLog[cp].ActiveTrans;
  DirtyPages := StableLog[cp].DirtyPages;
  max := LogSeqNo of most recent log entry in StableLog;
  for i := cp to max do
    case StableLog[i].ActionType:
      begin:
        losers += StableLog[i].TransId;
        losers[StableLog[i].TransId].LastSeqNo := nil;
      commit:
        losers −= StableLog[i].TransId;
      full−write:
        losers[StableLog[i].TransId].LastSeqNo := i;
    end /*case*/;
    if StableLog[i].ActionType = write or full−write
       and StableLog[i].PageNo not in DirtyPages
    then
      DirtyPages += StableLog[i].PageNo;
      DirtyPages[StableLog[i].PageNo].RedoSeqNo := i;
    end /*if*/;
  end /*for*/;
```



```
redo pass ( ):
  cp := MasterRecord.LastCP;
  SystemRedoLSN := min{cp.DirtyPages[p].RedoSeqNo};
  max := LogSeqNo of most recent log entry in StableLog;
  for i := SystemRedoLSN to max do
      if StableLog[i].ActionType = write or full−write
         and StableLog[i].TransId not in losers
      then
        pageno := StableLog[i].PageNo;
        if pageno in DirtyPages
           and i >= DirtyPages[pageno].RedoSeqNo
        then
          fetch (pageno);
          if DatabaseCache[pageno].PageSeqNo < i
          then
            read and write (pageno) according to StableLog[i].RedoInfo;
            DatabaseCache[pageno].PageSeqNo := i;
          else
            DirtyPages[pageno].RedoSeqNo :=
              DatabaseCache[pageno].PageSeqNo + 1;
          end /*if*/;
        end /*if*/;
      end /*if*/;
  end /*for*/;


undo pass ( ):
  while there exists t in losers
      such that losers[t].LastSeqNo <> nil
  do
     nexttrans = TransNo in losers
        such that losers[nexttrans].LastSeqNo =
        max {losers[x].LastSeqNo | x in losers};
    nextentry = losers[nexttrans].LastSeqNo;
    if StableLog[nextentry].ActionType = full−write
    then
      pageno = StableLog[nextentry].PageNo;
      fetch (pageno);
      if DatabaseCache[pageno].PageSeqNo >= nextentry
      then
        read and write (pageno) according to StableLog[nextentry].UndoInfo;
        DatabaseCache[pageno].PageSeqNo := nextentry−1;
      end /*if*/;
      losers[nexttrans].LastSeqNo :=
        StableLog[nextentry].PreviousSeqNo;
    end /*if*/;
  end /*while*/;
```