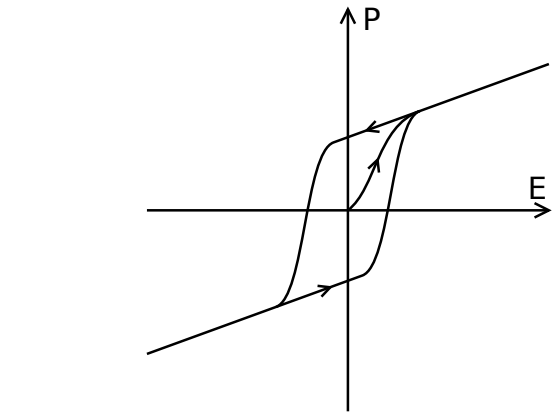
BS-Unterstützung für NVRAM

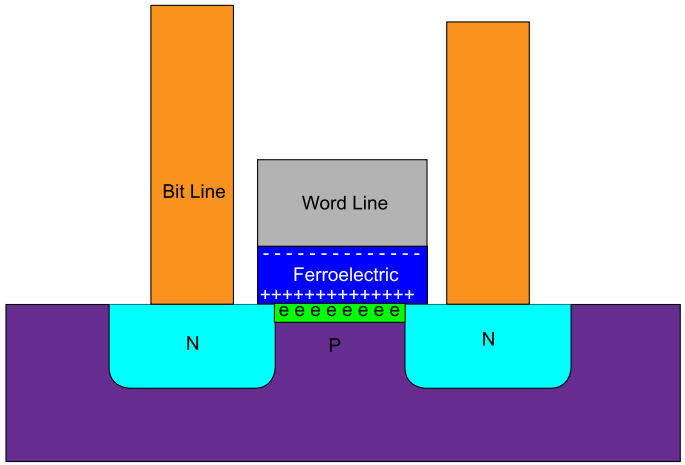
Timo Cramer

17. Oktober 2015

FRAM

- Ferroelectric RAM
  - Ferro- als Analogie zum Ferro-Magnetismus
  - kein Eisen, kein Magnetismus
- Schreiben intuitiv
- Lesen durch
  1. Schreibversuch
  2. Bemerken von Änderung
  3. Zustandswiederherstellung





[6, 7]

FRAM

- Ferroelectric RAM
  - Ferro- als Analogie zum Ferro-Magnetismus
  - kein Eisen, kein Magnetismus
- Schreiben intuitiv
- Lesen durch
  1. Schreibversuch
  2. Bemerken von Änderung
  3. Zustandswiederherstellung

\+ nicht flüchtig

\+ hohe Geschwindigkeit

\+ niedriger Energieverbrauch

\− teuer

\− geringe Dichte

Überlegungen

Was könnte sich ändern, wenn wir Systeme mit NVRAM haben?

- schnelle Wiederherstellung des Systemzustands
  - nach Ausfall
  - nach Reboot
- Ersetzen von langsameren, nicht-flüchtigen Speichermedien
- weniger „Sicherheitskopien“, die auf Massenspeicher geschrieben werden

nur-NVRAM-Systeme

Systeme, die in Zukunft nur mit NVRAM ausgestattet sind, sind denkbar.

- Verschmelzung zwischen Arbeits- und Massenspeicher
  - in-place Bearbeitung von Daten
  - im Extrem: Wegfall von Paging und getrennten Adressräumen[1]
- Problem: Medien sind dann nicht mehr einfach wechselbar

Arten der Wiederherstellung

Systemwiederherstellung kann in verschiedenen Ausformungen passieren

- kompletter Reboot
- wie Suspend & Resume
- auf Gerätetreiber-Ebene
- auf Applikations-Ebene

Eine komplette Wiederherstellung ist immer noch nicht (einfach) möglich

- flüchtige Register, Caches, etc.
- neu gestartete Peripherie nach Reboot/Ausfall

Probleme

- Speicherschutz wird wichtiger
  - Fortpflanzung und Persistenz von Fehlern
  - Neustart löst nicht mehr so viele Probleme ;)
- Security
  - Kryptographie-Schlüssel im Speicher
  - Stichwort „Cold-Boot-Attacke“

ACID

Häufigstes Ziel von Datenbanken und verteilten Systemen

Atomicy Jede Aktion wird ganz oder gar nicht ausgeführt

Consistency Jede Aktion bringt das System von einem konsistenten Zustand in den nächsten

Isolation Keine laufende Aktion beeinflusst eine andere laufende Aktion

Durability Das Ergebnis jeder erfolgreichen Aktion bleibt dauerhaft erhalten

Journaling

- Einsatz in Dateisystemen und Datenbanken
- Änderungen in Form von Transaktionen
- Speichern von Änderungsabsichten
- bei Systemausfall: schnelle Wiederherstellung eines konsistenten Zustands
- mögliche Operationen:
  - Undo – rückgängig
  - Redo – wiederholen

Journaling: Variationen[3]

- Writeback Mode
  - Nur Metadaten werden ins Journal geschrieben
  - Daten werden einfach auf die Platte geschrieben
  - \+ Dateisystem-Struktur bleibt konsistent
  - − Möglichkeit für inkonsistente Daten
- Ordered Mode
  - wie Writeback Mode
  - Metadaten werden **nach** den Daten ins Journal geschrieben
  - \+ keine Möglichkeit mehr für inkonsistente Daten
- Data Mode
  - Metadaten und Daten werden gejournaled
  - \+ noch mehr Schutz
  - − geringe Performance

Datenbanken: Shadow Pages[5]

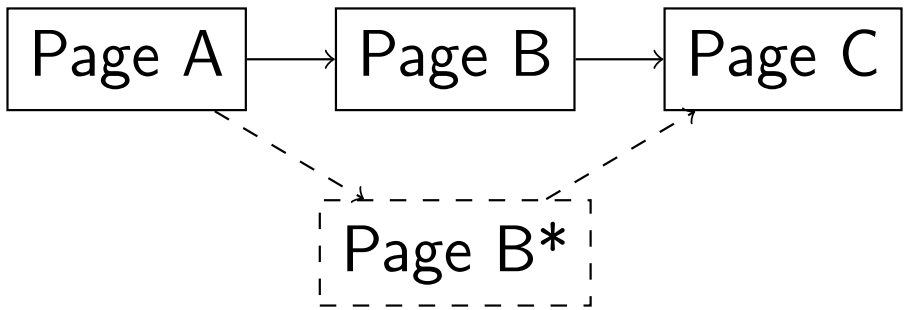
- Konsistenzwahrung durch Redundanz
- Änderungen passieren in einer temporären Page
- Bei Erfolg: Ersetzen der alten Page
- Bei Fehler: Entfernen der temporären Page

Beispiel: Änderungen an Page B

Datenbanken: Shadow Pages[5]

- Konsistenzwahrung durch Redundanz
- Änderungen passieren in einer temporären Page
- Bei Erfolg: Ersetzen der alten Page
- Bei Fehler: Entfernen der temporären Page

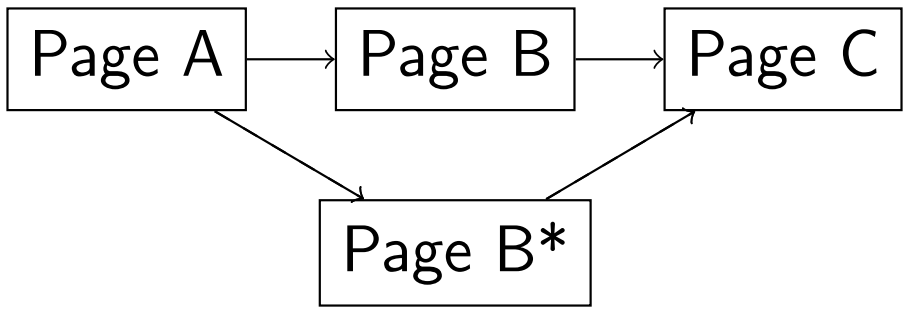
Beispiel: Änderungen an Page B



Datenbanken: Shadow Pages[5]

- Konsistenzwahrung durch Redundanz
- Änderungen passieren in einer temporären Page
- Bei Erfolg: Ersetzen der alten Page
- Bei Fehler: Entfernen der temporären Page

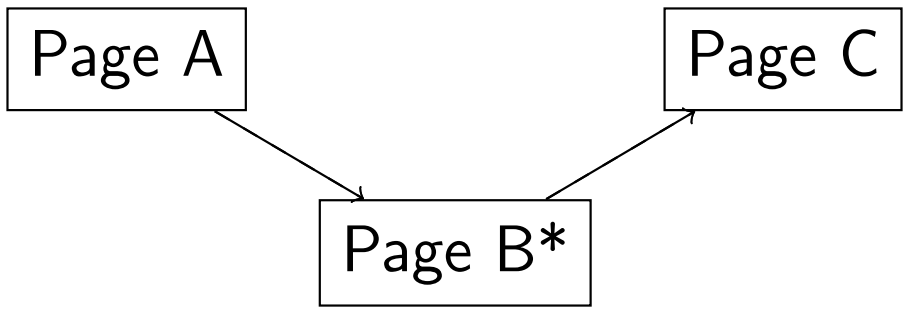
Beispiel: Änderungen an Page B



Datenbanken: Shadow Pages[5]

- Konsistenzwahrung durch Redundanz
- Änderungen passieren in einer temporären Page
- Bei Erfolg: Ersetzen der alten Page
- Bei Fehler: Entfernen der temporären Page

Beispiel: Änderungen an Page B



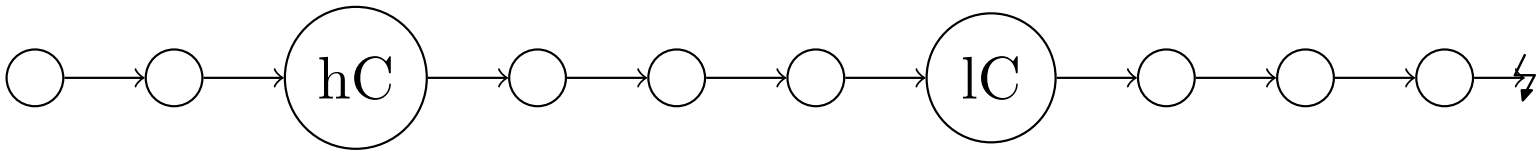
Datenbanken: Shadow Pages[5]

- Konsistenzwahrung durch Redundanz
- Änderungen passieren in einer temporären Page
- Bei Erfolg: Ersetzen der alten Page
- Bei Fehler: Entfernen der temporären Page

\+ schnelle Wiederherstellung

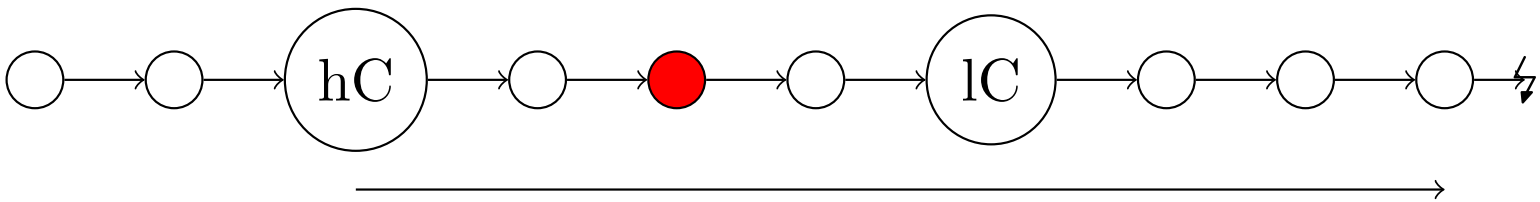
− schlechte Performance

Datenbanken: Checkpointing[5]



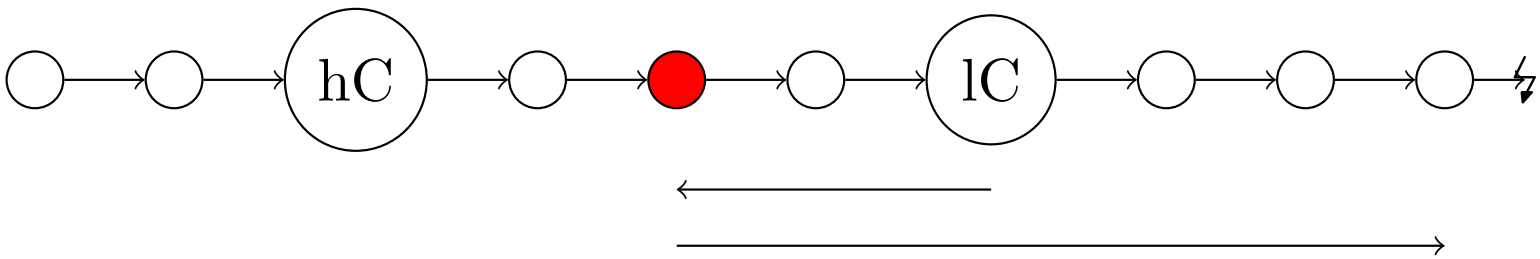
- Logging (Journaling) von allen Aktionen
- kein Durchgehen der kompletten Historie
- Wiederherstellung ab dem letzten Checkpoint
- Checkpoint-Arten
  - heavyweight Checkpoints
  - lightweight Checkpoints („fuzzy Checkpoints“)

Datenbanken: Checkpointing[5]



- Logging (Journaling) von allen Aktionen
- kein Durchgehen der kompletten Historie
- Wiederherstellung ab dem letzten Checkpoint
- Checkpoint-Arten
  - heavyweight Checkpoints
  - lightweight Checkpoints („fuzzy Checkpoints“)

Datenbanken: Checkpointing[5]



- Logging (Journaling) von allen Aktionen
- kein Durchgehen der kompletten Historie
- Wiederherstellung ab dem letzten Checkpoint
- Checkpoint-Arten
  - heavyweight Checkpoints
  - lightweight Checkpoints („fuzzy Checkpoints“)

Fine-Grained Fault Tolerance[2]

. . . bietet feinere Wiederherstellung von Linux-Gerätetreibern

- Neustart dauert zu lang
- Techniken:
  - Isolation
  - Checkpoints
- Anforderungen:
  - Schnelligkeit
  - Allgemeinheit
  - Einsetzbarkeit bei Multithreading

Fine-Grained Fault Tolerance[2]

Isolation

- Transactional Execution
- Memory safety and fault detection
- Synchronization

. . . durch Code-Analyse und -Generierung zur Übersetzungszeit

- minimale Kopie der benutzten Daten
- Erstellung der Kopien an *entry points*
- Zurückschreiben bei Erfolg

Fine-Grained Fault Tolerance[2]

Checkpoints

- Speicher-basierter Ansatz nicht sinnvoll
  - Doppelbelegung von Adressen
  - Seiteneffekte
- Wiederverwendung von Power Management
  - Suspend & Resume
  - z. B. Speichern von PCI-Zustand
- Kein komplettes Zustandsabbild *des Gerätes*
  - kein einfacher Memory-Dump
  - Speichern von Konfiguration und Registerinhalten
  - geteilte Buffer werden bei Wiederherstellung z. B. neu initialisiert

Whole System Persistence[4]

. . . ist ein Ansatz z. B. für Hauptspeicherdatenbanken

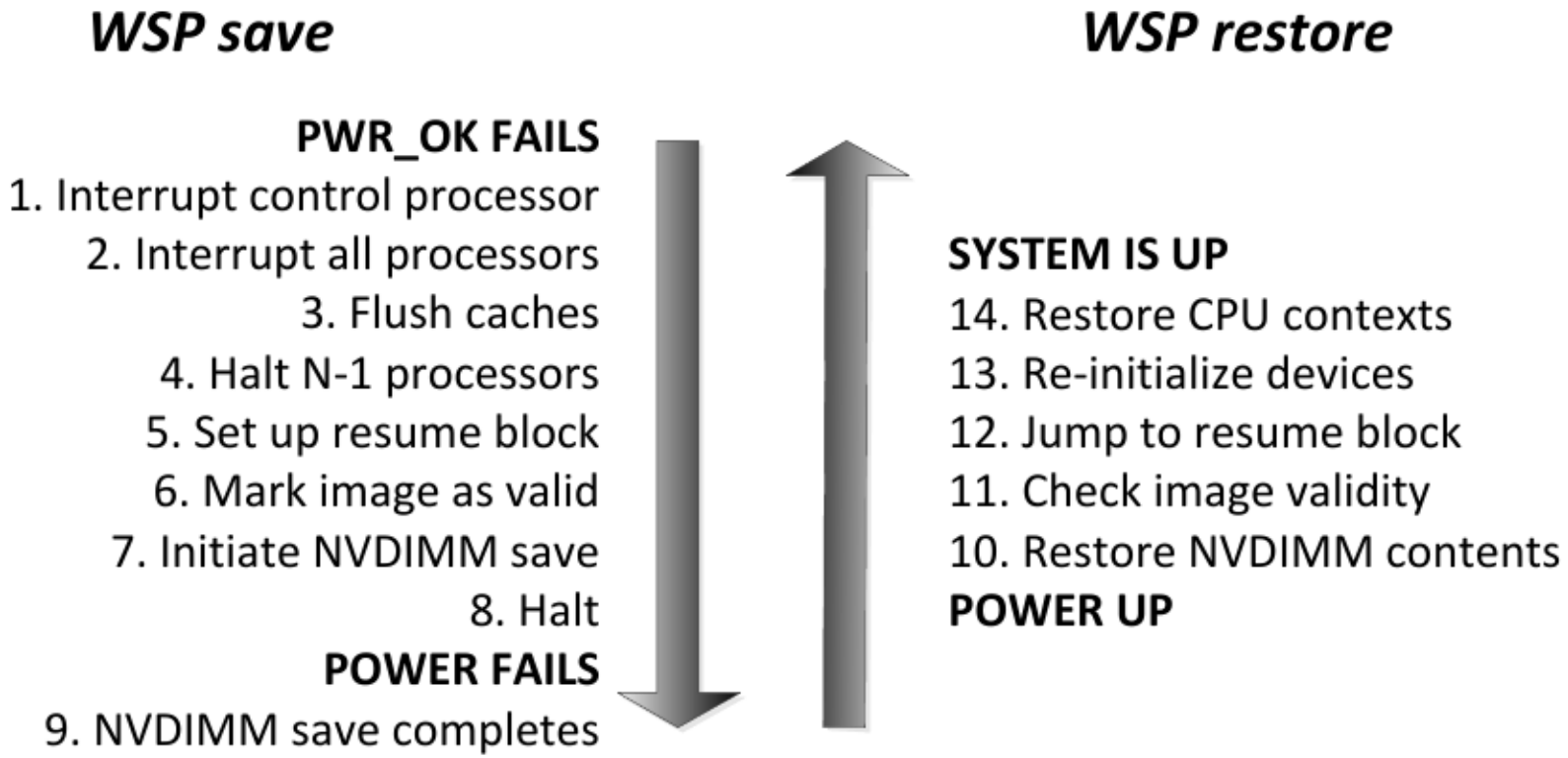
- mehrere 100 GB Hauptspeicher
- langwierige Wiederherstellung aus dem Massenspeicher
- Hauptspeicher ist komplett nicht-flüchtig

Idee:

- Betriebssystem speichert bei Stromausfall den gesamten Zustand
- Speichern dauert etwa 5 ms, Stromversorgung liefert noch 10–300 ms Strom

Whole System Persistence[4]

- Überwachung der Stromversorgung durch einen Arduino
- Interrupt-Signal über serielle Schnittstelle



Zusammenfassung

- NVRAM hilft uns, persistente Systeme zu erstellen
- komplette Wiederherstellung ist immer noch nicht einfach möglich
- zustandsbasierte vs. speicherbasierte Sichtweise

Empfehlungen

- Whole System Persistence
  - beeinflusst den Rest der Programmierung nicht
  - Hardware/Energiemodell muss vorhanden sein
- „Sicherheitskopien“ mit Shadow Objects
  - wir haben ein System mit flüchtigem und nicht-flüchtigem RAM
  - Arbeitskopie im SRAM, dauerhafte Kopie im FRAM

Quellen I


Katelin Bailey, Luis Ceze, Steven D. Gribble, and Henry M. Levy.
Operating System Implications of Fast, Cheap, Non-Volatile Memory.
2011.


Asim Kadav, Matthew J. Renzelmann, and Michael M. Swift.
Fine-Grained Fault Tolerance using Device Checkpoints.
2013.

Quellen II


M. Tim Jones.
Anatomy of Linux journaling file systems.
http://www.ibm.com/developerworks/library/l-journaling-filesystems/, 2008-06-04.


Dushyanth Narayanan and Orion Hodson.
Whole-System Persistence.
2012.


Jens Teubner.
Vorlesungsfolien zu AIDB, WS 14/15.

Quellen III


Wikipedia User Bigly.
Ferroelectric polarisation.svg.
https://de.wikipedia.org/wiki/Datei:Ferroelectric_polarisation.svg, 2007-12-03.


Wikipedia User Cyferz.
1T FeRAM cell structure.svg.
https://en.wikipedia.org/wiki/File:1T_FeRAM_cell_structure.svg, 2007-07-09.