

# Betriebssystemtechnik

*Adressräume: Trennung, Zugriff, Schutz*

## XII. Spezialfälle: Virtuell nichtflüchtiger Hauptspeicher

---

SS 2026

Wolfgang Schröder-Preikschat / Volkmar Sieh



**Lehrstuhl für Informatik 4**  
Systemsoftware



Friedrich-Alexander-Universität  
Technische Fakultät

## Einleitung

## Grundlagen

Datenkonsistenz

Ausfalltoleranz

Zugriffslatenz

## Fallstudien

Rauscharmes System

Kaschierendes System

Resilientes System

## Zusammenfassung

## Nichtflüchtiger (*non-volatile*) RAM

---

<sup>1</sup>Nicht zu verwechseln mit \*ROM (*read-only memory*).

- Art von **Hauptspeicher**, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert<sup>1</sup>
  - byteadressierbarer nichtflüchtiger Speicher (*non-volatile memory*, NVM)
  - anders als ein Halbleiterlaufwerk wie SSD, das blockorientiert arbeitet

---

<sup>1</sup>Nicht zu verwechseln mit \*ROM (*read-only memory*).

- Art von **Hauptspeicher**, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert<sup>1</sup>
  
- Hochskalieren der Speicherkapazität auf DRAM-Basis kann sowohl hinsichtlich Leistung als auch Kosten unerschwinglich teuer sein
  - DRAM-Auffrischleistung skaliert proportional mit der Speicherkapazität
  - außerdem ist weiteres Herunterskalieren der Kondensatorgröße schwierig

---

<sup>1</sup>Nicht zu verwechseln mit \*ROM (*read-only memory*).

- Art von **Hauptspeicher**, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert<sup>1</sup>
  
- Hochskalieren der Speicherkapazität auf DRAM-Basis kann sowohl hinsichtlich Leistung als auch Kosten unerschwinglich teuer sein
  
- seine hohe Dichte, geringe Standby-Leistung und niedrige Kosten pro Bit machen NVM zur vielversprechenden Alternative zu DRAM
  - gleichwohl kann die Zugriffslatenz 3- bis 20-fach höher als bei DRAM sein
  - zudem niedrigere Bandbreite und asymmetrische Lese-/Schreibleistung↪ Punkte, die einer Eignung als primärer Hauptspeicher noch widersprechen

---

<sup>1</sup>Nicht zu verwechseln mit \*ROM (*read-only memory*).

- Art von **Hauptspeicher**, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert<sup>1</sup>

Auslaufmodell DRAM, es kommt wohl „Kernspeicher“ zurück — [10]:

- Hochskalieren der Speicherkapazität auf DRAM-Basis kann sowohl hinsichtlich Leistung als auch Kosten unerschwinglich teuer sein
  
- seine hohe Dichte, geringe Standby-Leistung und niedrige Kosten pro Bit machen NVM zur vielversprechenden Alternative zu DRAM

---

<sup>1</sup>Nicht zu verwechseln mit \*ROM (*read-only memory*).



- hinter „disruptive Technologie“ stehen „bahnbrechende Innovationen“

- hinter „disruptive Technologie“ stehen „bahnbrechende Innovationen“ *[...] die zunächst nicht den Bedürfnissen der Mainstream-Kunden entsprechen und nur kleine oder aufstrebende Märkte ansprechen; solche, die zumindest am Anfang von Bestandskunden nicht geschätzt werden; technologische Veränderungen, die etablierten Unternehmen schaden. [2]*

- hinter „disruptive Technologie“ stehen „bahnbrechende Innovationen“

*[...] die etablierte Technologien obsolet machen und daher den Wert der Investitionen zerstören, die etablierte Unternehmen in diese Technologien getätigt haben. [3]*



- bahnbrechend und zugleich zerstörend in Bezug auf Hauptspeicher
  - große Kapazität, geringe Bereitschaftsleistung, ..., **persistent**

- bahnbrechend und zugleich zerstörend in Bezug auf Hauptspeicher
  - große Kapazität, geringe Bereitschaftsleistung, ..., **persistent**
  - nicht abwärtskompatibel für Altsoftware, die „NVM-only“ laufen muss
    - sollte DRAM vollständig durch NVRAM ersetzt werden

Einleitung

Grundlagen

Datenkonsistenz

Ausfalltoleranz

Zugriffslatenz

Fallstudien

Rauscharmes System

Kaschierendes System

Resilientes System

Zusammenfassung

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

```
1 typedef struct date {
2     unsigned char day;
3     unsigned char month;
4     unsigned short year;
5 } date_t;
6 extern date_t retire;
7 retire.day = 30;
8 retire.month = 9;
9 retire.year = 2023;
```

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

```
1  typedef struct date {                6  extern date_t retire;
2      unsigned char day;
3      unsigned char month;            7  retire.day = 30;
4      unsigned short year;           8  retire.month = 9;
5  } date_t;                          9  retire.year = 2023;
```

- angenommen, das Programm kommt direkt im NVRAM zur Ausführung

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

```
1 typedef struct date {                6 extern date_t retire;
2     unsigned char day;
3     unsigned char month;            7     retire.day = 30;
4     unsigned short year;           8     retire.month = 9;
5 } date_t;                          9     retire.year = 2023;
```

- angenommen, das Programm kommt direkt im NVRAM zur Ausführung
- weiter angenommen, im Zeilenbereich [7, 9] kommt es zum Stromausfall<sup>2</sup>

---

<sup>2</sup>...und der gcc macht daraus kein „movl \$132581662, retire“ 😊

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

```
1 typedef struct date {                6 extern date_t retire;
2     unsigned char day;
3     unsigned char month;            7     retire.day = 30;
4     unsigned short year;           8     retire.month = 9;
5 } date_t;                          9     retire.year = 2023;
```

- angenommen, das Programm kommt direkt im NVRAM zur Ausführung
- weiter angenommen, im Zeilenbereich [7, 9] kommt es zum Stromausfall<sup>2</sup>
  - was lässt sich dann zur **Konsistenz** der Variablen retire festhalten?

---

<sup>2</sup>...und der gcc macht daraus kein „movl \$132581662, retire“ 😊

*Stromausfälle in Kombination mit NVM zum Schutz des Rechenzustands bewirken Kontrollflüsse, die einen sequentiellen Prozess unerwartet in einen nichtsequentiellen Prozess verwandeln können.*

- ein Programm muss sich mit seinem eigenen Zustand aus früheren unterbrochenen Läufen auseinandersetzen [11]

```
1  typedef struct date {                6  extern date_t retire;
2      unsigned char day;
3      unsigned char month;            7      retire.day = 30;
4      unsigned short year;           8      retire.month = 9;
5  } date_t;                          9      retire.year = 2023;
```

- angenommen, das Programm kommt direkt im NVRAM zur Ausführung
- weiter angenommen, im Zeilenbereich [7, 9] kommt es zum Stromausfall<sup>2</sup>
  - was lässt sich dann zur **Konsistenz** der Variablen retire festhalten?
  - was wäre der Unterschied, sollte retire keine **persistente Variable** sein?

---

<sup>2</sup>...und der gcc macht daraus kein „movl \$132581662, retire“ 😊

- sei auch folgender Code für den „NVM-only“ Betrieb vorgesehen:

```
1 typedef struct chain {
2     struct chain *link;
3 } chain_t;
4
5 typedef struct queue {
6     chain_t head;
7     chain_t *tail;
8 } queue_t; /* init: head = NULL; tail = &head */
9
10 void enqueue(queue_t *this, chain_t *item) {
11     item->link = 0;
12     this->tail->link = item;
13     this->tail = item;
14 }
```

- sei auch folgender Code für den „NVM-only“ Betrieb vorgesehen:

```
1 typedef struct chain {
2     struct chain *link;
3 } chain_t;
4
5 typedef struct queue {
6     chain_t head;
7     chain_t *tail;
8 } queue_t; /* init: head = NULL; tail = &head */
9
10 void enqueue(queue_t *this, chain_t *item) {
11     item->link = 0;
12     this->tail->link = item;
13     this->tail = item;
14 }
```

- angenommen, im Zeilenbereich [9, 11] wird der Prozess abgefangen (*trap*)
  - weil einer der Zeiger (*this*, *item*, *tail*) ungültig ist (*access violation*)

- sei auch folgender Code für den „NVM-only“ Betrieb vorgesehen:

```
1 typedef struct chain {
2     struct chain *link;
3 } chain_t;
4
5 typedef struct queue {
6     chain_t head;
7     chain_t *tail;
8 } queue_t; /* init: head = NULL; tail = &head */
9
10 void enqueue(queue_t *this, chain_t *item) {
11     item->link = 0;
12     this->tail->link = item;
13     this->tail = item;
14 }
```

- angenommen, im Zeilenbereich [9, 11] wird der Prozess abgefangen (*trap*)
  - weil einer der Zeiger (*this*, *item*, *tail*) ungültig ist (*access violation*)
  - weil die betreffende Seite ausgelagert ist (*page fault*)

- sei auch folgender Code für den „NVM-only“ Betrieb vorgesehen:

```
1 typedef struct chain {
2     struct chain *link;
3 } chain_t;
4
5 typedef struct queue {
6     chain_t head;
7     chain_t *tail;
8 } queue_t; /* init: head = NULL; tail = &head */
9
10 void enqueue(queue_t *this, chain_t *item) {
11     item->link = 0;
12     this->tail->link = item;
13     this->tail = item;
14 }
```



- angenommen, im Zeilenbereich [9, 11] wird der Prozess abgefangen (*trap*)
  - weil einer der Zeiger (*this*, *item*, *tail*) ungültig ist (*access violation*)
  - weil die betreffende Seite ausgelagert ist (*page fault*) — und der Faden, der `enqueue()` ausführt, von einem „befeundeten“ Faden terminiert wird

---

- eine instantane Aktualisierung logisch zusammenhängender, mehrere Speicherworte umfassender Daten ist nicht garantiert
  - potentiell inkonsistente Daten können die Folge sein
    - ein Datenanteil ist bereits persistent, der andere Datenanteil ist verloren
    - letzterer fiel in die hypothetische Zukunft, die aber nicht stattfand
  - solche Aktualisierungen müssten daher eine **Transaktion**<sup>3</sup> bilden

---

<sup>3</sup>Hinterlässt den Datenbestand nach fehlerfreier und vollständiger Ausführung einer Aktualisierung in einem konsistenten Zustand.

- eine instantane Aktualisierung logisch zusammenhängender, mehrere Speicherworte umfassender Daten ist nicht garantiert
  
  - die **Konsistenzprobleme** stammen von Verletzungen der erwarteten Atomizität und Ordnungseigenschaften
    - nichtsequentielle Programme beugen gemeinhin solchen Problemen vor
      - durch **Synchronisation** der kritischen gleichzeitigen Prozesse
    - für gewöhnliche sequentielle Programme war dies bisher unnötig
      - da der betreffende Prozess beim Abbruch im RAM „spurlos“ bleibt
-

- eine instantane Aktualisierung logisch zusammenhängender, mehrere Speicherworte umfassender Daten ist nicht garantiert
  
  - die **Konsistenzprobleme** stammen von Verletzungen der erwarteten Atomizität und Ordnungseigenschaften
  
  - um auch **Altsoftware** (*legacy code*) gegen NVM zu wappnen, ist es erforderlich, Prozesse idempotent geschehen zu lassen
    - unter allen Umständen den flüchtigen Zustand noch sichern können...
-

- eine instantane Aktualisierung logisch zusammenhängender, mehrere Speicherworte umfassender Daten ist nicht garantiert
  
  - die **Konsistenzprobleme** stammen von Verletzungen der erwarteten Atomizität und Ordnungseigenschaften
  
  - um auch **Altsoftware** (*legacy code*) gegen NVM zu wappnen, ist es erforderlich, Prozesse idempotent geschehen zu lassen
-

- bei plötzlichem **Stromausfall** noch in der Lage sein, Maßnahmen zu ergreifen, um mögliche Dateninkonsistenzen abzuwenden

- bei plötzlichem **Stromausfall** noch in der Lage sein, Maßnahmen zu ergreifen, um mögliche Dateninkonsistenzen abzuwenden
  - den **Übergangszustand** (*transient state*) der CPU in den NVM sichern
    - das heißt, die in den Registern und dem Cache enthaltenen Daten
  - wobei die **Restenergie** aus der Systemstromversorgung verwendet wird
    - „flush on fail“ [8] im **Restenergiefenster** von Standardnetzteilen

- bei plötzlichem **Stromausfall** noch in der Lage sein, Maßnahmen zu ergreifen, um mögliche Dateninkonsistenzen abzuwenden
  
  - die Ausfallsicherung ist begrenzt in Raum und Zeit, nämlich durch die Registeranzahl, Cachegröße und Breite des Restenergiefensters
    - sei der Umfang des Übergangszustands klein genug für das Zeitfenster
    - dann ist sicherzustellen, dass die **Sicherungsprozedur** rechtzeitig startet
      - beim „*power failure trap*“ sofort: ein Trap kann nicht gesperrt werden 😊
      - beim „*power failure interrupt*“ (PFI) verzögert: **Unterbrechungssperre** 😞
-

- bei plötzlichem **Stromausfall** noch in der Lage sein, Maßnahmen zu ergreifen, um mögliche Dateninkonsistenzen abzuwenden
  
- die Ausfallsicherung ist begrenzt in Raum und Zeit, nämlich durch die Registeranzahl, Cachegröße und Breite des Restenergiefensters
  
- Stand heutiger Technik bei den Prozessoren ist der PFI, angefordert durch eine **unterbrechungsfreie Stromversorgung** (USV)
  - für die Sicherung gilt: **Auslösezeit** + **WCET**<sup>4</sup> < **harter Termin**

---

<sup>4</sup>worst case execution time

- bei plötzlichem **Stromausfall** noch in der Lage sein, Maßnahmen zu ergreifen, um mögliche Dateninkonsistenzen abzuwenden
  
- die Ausfallsicherung ist begrenzt in Raum und Zeit, nämlich durch die Registeranzahl, Cachegröße und Breite des Restenergiefensters
  
- Stand heutiger Technik bei den Prozessoren ist der PFI, angefordert durch eine **unterbrechungsfreie Stromversorgung** (USV)
  - für die Sicherung gilt: **Auslösezeit + WCET<sup>4</sup> < harter Termin**
  - ↪ keine Maßnahme im Betriebssystem darf diese Bedingung verletzen!

---

<sup>4</sup>worst case execution time

- Spezifikationen für die durch eine Stromversorgungseinheit mögliche „Fensterbreite“ (in Zeit) sind ausstattungsabhängig:
  - 10–400ms**     ■ Reserve bei Standardnetzteilen eines PC [8]
  - 25–30s**     ■ Reserve einer USV mit Schwungrad (*flywheel*, [1])
  - 30s–12min**   ■ Anlaufzeit eines Diesel-Notstromaggregats
  - Überbrückungszeit für batteriebasierte Ausfallprävention

- Spezifikationen für die durch eine Stromversorgungseinheit mögliche „Fensterbreite“ (in Zeit) sind ausstattungsabhängig:
  - 10–400ms**     ■ Reserve bei Standardnetzteilen eines PC [8]
  - 25–30s**     ■ Reserve einer USV mit Schwungrad (*flywheel*, [1])
  - 30s–12min**   ■ Anlaufzeit eines Diesel-Notstromaggregats
  - Überbrückungszeit für batteriebasierte Ausfallprävention
- kann der Übergangszustand der CPU mit der bei Standardnetzteilen verfügbaren Reserve gesichert werden, ist das System sicher

- Spezifikationen für die durch eine Stromversorgungseinheit mögliche „Fensterbreite“ (in Zeit) sind ausstattungsabhängig:
  - 10–400ms**    ■ Reserve bei Standardnetzteilen eines PC [8]
  - 25–30s**     ■ Reserve einer USV mit Schwungrad (*flywheel*, [1])
  - 30s–12min** ■ Anlaufzeit eines Diesel-Notstromaggregats
  - Überbrückungszeit für batteriebasierte Ausfallprävention
- kann der Übergangszustand der CPU mit der bei Standardnetzteilen verfügbaren Reserve gesichert werden, ist das System sicher
  - die **Sicherungsprozedur** wird in endlicher Zeit durchlaufen können
    - Größe und Anzahl der zu sichernden Prozessorregister ist begrenzt
    - Größe und Anzahl der zu sichernden Cachezeilen ist begrenzt
  - ↪ die **maximale Ausführungszeit** (WCET) ist bestimmbar: **statische Analyse**

- Spezifikationen für die durch eine Stromversorgungseinheit mögliche „Fensterbreite“ (in Zeit) sind ausstattungsabhängig:

**10–400ms**

- Reserve bei Standardnetzteilen eines PC [8]

**25–30s**

- Reserve einer USV mit Schwungrad (*flywheel*, [1])

**30s–12min**

- Anlaufzeit eines Diesel-Notstromaggregats
- Überbrückungszeit für batteriebasierte Ausfallprävention

- kann der Übergangszustand der CPU mit der bei Standardnetzteilen verfügbaren Reserve gesichert werden, ist das System sicher

- kritisch ist ihre Auslösezeit und dass der PFI rechtzeitig behandelt wird
  - betrifft die **Unterbrechungspriorität** des PFI und **Unterbrechungssperren**
  - beides bestimmt die **Latenz**, die Verzögerungszeit der Zustandssicherung
  - ↪ erstere muss die höchste Stufe sein, die WCET letzterer muss bekannt sein



- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperrern** relevant

- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperrern** relevant
  - beispielsweise CFS (*completely fair scheduling, fair.c*) in Linux:

```
...  
double_lock_irq()  
  ↳ spin_lock_irq()  
    ↳ ...  
      ↳ local_irq_disable()
```

- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperr**en relevant
  - beispielsweise CFS (*completely fair scheduling, fair.c*) in Linux:

...

```
double_lock_irq()
↳ spin_lock_irq()
  ↳ ...
    ↳ local_irq_disable()
```

- charakteristisches Muster, um möglichen Verklemmungen vorzubeugen
  - wenn die Unterbrechungsbehandlung dieselben Anweisungen ausführen muss
  - wegen präemptivem Prozessscheduling und blockierender Synchronisation

- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperr**en relevant
  - beispielsweise CFS (*completely fair scheduling, fair.c*) in Linux:

...

```
double_lock_irq()  
  ↳ spin_lock_irq()  
    ↳ ...  
      ↳ local_irq_disable()
```

- charakteristisches Muster, um möglichen Verklemmungen vorzubeugen
  
- ein nichtblockierend synchronisiertes Betriebssystem ist unbehaftet damit

- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperr**en relevant
  - beispielsweise CFS (*completely fair scheduling, fair.c*) in Linux:

```
...
double_lock_irq()
  ↳ spin_lock_irq()
    ↳ ...
      ↳ local_irq_disable()
```

*Solche Abschnitte sind zu lokalisieren, ihre jeweilige WCET ist zu bestimmen und gegebenenfalls ist der Kode umzustellen!*

- charakteristisches Muster, um möglichen Verklemmungen vorzubeugen

- Maßnahme, durch die ein **kritischer Abschnitt** vor Überlappung mit einer **Unterbrechungsbehandlungsroutine** bewahrt wird
  - obwohl nur CPU-lokal signifikant, auch für **Umlaufsperr**en relevant
  - beispielsweise CFS (*completely fair scheduling, fair.c*) in Linux:

```
...
double_lock_irq()
  ↳ spin_lock_irq()
    ↳ ...
      ↳ local_irq_disable()
```

*Solche Abschnitte sind zu lokalisieren, ihre jeweilige WCET ist zu bestimmen und gegebenenfalls ist der Code umzustellen!*

- eine sehr ähnliche Problematik zeigt sich im Zusammenhang mit dem Scheduling virtueller Maschinen: „*lock holder preemption*“ [13]
  - obgleich Sperren eines IRQ hier für gewöhnlich nicht die erste Wahl ist...



- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab
  - das geschieht weitestgehend analog zum gewöhnlichen **Prozesswechsel**
    - den Prozess wiederaufnehmen (*resume*), wie es auch normalerweise geschieht
  - jedoch wechselt ein Prozess mit allem in die „Ohnmacht“ (*blackout*)
    - aus der er beim Neuanlauf (*restart*) des Systems wieder erwachen wird

- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab
  
- sei die letzte Maßnahme der Sicherungsprozedur, den Prozessorstatus des beim Stromausfall laufenden Prozesses in den NVM zu schreiben

```
1  goodbye:
2      movl    4(%esp), %eax
3      movl    %edi, 0(%eax)
4      movl    %esi, 4(%eax)
5      movl    %ebp, 8(%eax)
6      movl    %esp, 12(%eax)
7      movl    %ebx, 16(%eax)
8      movl    $welcome, 20(%eax)
9      hlt
```

- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab

- sei die letzte Maßnahme der Sicherungsprozedur, den Prozessorstatus des beim Stromausfall laufenden Prozesses in den NVM zu schreiben

```
1  goodbye:
2      movl   4(%esp), %eax
3      movl   %edi, 0(%eax)
4      movl   %esi, 4(%eax)
5      movl   %ebp, 8(%eax)
6      movl   %esp, 12(%eax)
7      movl   %ebx, 16(%eax)
8      movl   $welcome, 20(%eax)
9      hlt
```

- der Prozess sichert seinen Zustand (3–7)
  - **nichtflüchtige Register** der CPU puffern
  - der Puffer liegt im NVRAM
  - die Pufferadresse ist aktueller Parameter

- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab

- sei die letzte Maßnahme der Sicherungsprozedur, den Prozessorstatus des beim Stromausfall laufenden Prozesses in den NVM zu schreiben

```
1  goodbye:
2      movl   4(%esp), %eax
3      movl   %edi, 0(%eax)
4      movl   %esi, 4(%eax)
5      movl   %ebp, 8(%eax)
6      movl   %esp, 12(%eax)
7      movl   %ebx, 16(%eax)
8      movl   $welcome, 20(%eax)
9      hlt
```

- der Prozess sichert seinen Zustand (3–7)
  - **nichtflüchtige Register** der CPU puffern
  - der Puffer liegt im NVRAM
  - die Pufferadresse ist aktueller Parameter
- er hinterlässt seine Fortsetzungsadresse (8)
  - welcome, ebenfalls im NVRAM gepuffert

- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab

- sei die letzte Maßnahme der Sicherungsprozedur, den Prozessorstatus des beim Stromausfall laufenden Prozesses in den NVM zu schreiben

```
1  goodbye:
2      movl   4(%esp), %eax
3      movl   %edi, 0(%eax)
4      movl   %esi, 4(%eax)
5      movl   %ebp, 8(%eax)
6      movl   %esp, 12(%eax)
7      movl   %ebx, 16(%eax)
8      movl   $welcome, 20(%eax)
9      hlt
```

- der Prozess sichert seinen Zustand (3–7)
  - **nichtflüchtige Register** der CPU puffern
  - der Puffer liegt im NVRAM
  - die Pufferadresse ist aktueller Parameter
- er hinterlässt seine Fortsetzungsadresse (8)
  - welcome, ebenfalls im NVRAM gepuffert
- abschließend stoppt er seinen Prozessor (9) – fällt in „Ohnmacht“...

- ein unterbrochener Prozess nimmt seinen Ablauf an der Stelle wieder auf, an der er aussetzen musste und den Prozessor abgab
  
- sei die letzte Maßnahme der Sicherungsprozedur, den Prozessorstatus des beim Stromausfall laufenden Prozesses in den NVM zu schreiben

```
1  goodbye:
2     movl   4(%esp), %eax
3     movl   %edi, 0(%eax)
4     movl   %esi, 4(%eax)
5     movl   %ebp, 8(%eax)
6     movl   %esp, 12(%eax)
7     movl   %ebx, 16(%eax)
8     movl   $welcome, 20(%eax)
9     hlt
```

↪ **der Prozess muss innerhalb des Restenergiefensters `hlt` erreichen**



- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)
  - das dort befindliche Programm initialisiert das System, fährt es hoch
  - zusätzlich prüft es auf etwaige **Hinterlassenschaft** eines früheren Laufs
    - beispielsweise die Adresse des Sicherungspuffers der Sicherungsprozedur
    - die Zeigervariable dafür liegt selbst im NVM, wird nach Prüfung gelöscht

- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)
  
- im Falle eines vorangegangenen Stromausfalls werden, soweit nötig, Wiederherstellungsmaßnahmen für das System getroffen
  - zuletzt auch die **Wiederaufnahme** des vorher unterbrochenen Prozesses
    - an der im Sicherungspuffer hinterlassenen **Fortsetzungsadresse**

- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)
  
- im Falle eines vorangegangenen Stromausfalls werden, soweit nötig, Wiederherstellungsmaßnahmen für das System getroffen
  - zuletzt auch die **Wiederaufnahme** des vorher unterbrochenen Prozesses
    - an der im Sicherungspuffer hinterlassenen **Fortsetzungsadresse**
  - das heißt, welcome anspringen und den Prozess wieder in Gang setzen

```
1  welcome:
2      movl 4(%esp), %eax
3      movl 0(%eax), %edi
4      movl 4(%eax), %esi
5      movl 8(%eax), %ebp
6      movl 16(%eax), %ebx
7      movl 12(%eax), %esp
8      ret
```

- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)

- im Falle eines vorangegangenen Stromausfalls werden, soweit nötig, Wiederherstellungsmaßnahmen für das System getroffen

```
1  welcome:
2      movl  4(%esp), %eax
3      movl  0(%eax), %edi
4      movl  4(%eax), %esi
5      movl  8(%eax), %ebp
6      movl 16(%eax), %ebx
7      movl 12(%eax), %esp
8      ret
```

- den Prozesszustand übernehmen (3–6)
  - **nichtflüchtige Register** der CPU definieren
  - die Werte dem Sicherungspuffer entnehmen
  - die Pufferadresse ist aktueller Parameter

- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)

- im Falle eines vorangegangenen Stromausfalls werden, soweit nötig, Wiederherstellungsmaßnahmen für das System getroffen

```
1  welcome:
2      movl   4(%esp), %eax
3      movl   0(%eax), %edi
4      movl   4(%eax), %esi
5      movl   8(%eax), %ebp
6      movl   16(%eax), %ebx
7      movl   12(%eax), %esp
8      ret
```

- den Prozesszustand übernehmen (3–6)
  - **nichtflüchtige Register** der CPU definieren
  - die Werte dem Sicherungspuffer entnehmen
  - die Pufferadresse ist aktueller Parameter

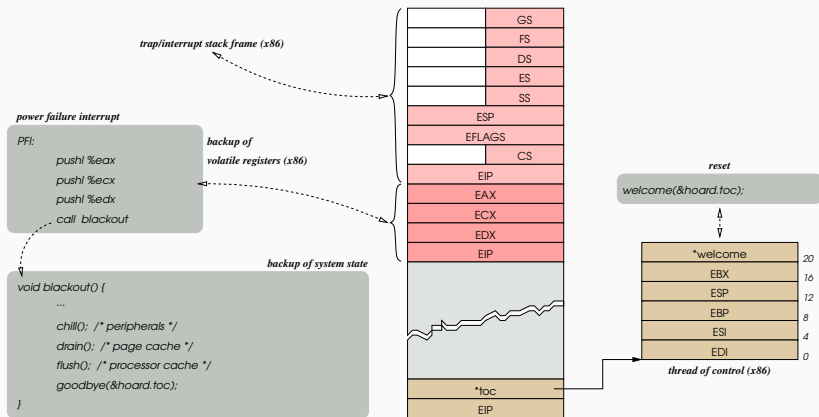
- den Laufzeitstapel umschalten (7) und zum Prozess zurückkehren (8)

- der Neuanlauf des Systems beginnt in gewöhnlicher **Grundstellung** (*reset*) des Prozessors, mit fester **Einsprungsadresse** (*reset vector*)
  
- im Falle eines vorangegangenen Stromausfalls werden, soweit nötig, Wiederherstellungsmaßnahmen für das System getroffen

```
1  welcome:
2      movl  4(%esp), %eax
3      movl  0(%eax), %edi
4      movl  4(%eax), %esi
5      movl  8(%eax), %ebp
6      movl 16(%eax), %ebx
7      movl 12(%eax), %esp
8      ret
```

↪ **der Prozess setzt genau dort fort, wo er sich verabschiedet hatte**

# Hinterlassenschaft beim Stromausfall



- Ausschnitt des Prozessorstatus' (x86) nach Verabschiedung des durch den Stromausfall unterbrochenen Prozesses
  - der Übergangszustand des Systems wurde in NVRAM persistent gemacht

- der Fall „**NVM-only**“ soll bedeuten, dass die Maschinenprogramme und das Betriebssystem im NVRAM liegend ausgeführt werden
  - PFI-basierte Konsistenzwahrung kaschiert die Untiefen von NVM
  - lediglich die Sicherungsprozedur ist transaktional zu programmieren

- der Fall „**NVM-only**“ soll bedeuten, dass die Maschinenprogramme und das Betriebssystem im NVRAM liegend ausgeführt werden
- die damit erreichbare vollständige **Persistenz** aller Programme geht jedoch auf Kosten der **Performanz** der entsprechenden Prozesse
  - die Zugriffslatenz kann 3- bis 20-fach höher sein als bei DRAM
  - nur mit **Resilienz** als Maß aller Dinge könnte dies tolerierbar sein

- der Fall „**NVM-only**“ soll bedeuten, dass die Maschinenprogramme und das Betriebssystem im NVRAM liegend ausgeführt werden
- die damit erreichbare vollständige **Persistenz** aller Programme geht jedoch auf Kosten der **Performanz** der entsprechenden Prozesse
- der Ansatz ist nun, den NVRAM zu virtualisieren und durch DRAM als schnelleren Zwischenspeicher die Zugriffslatenz zu verkürzen
  - ein DRAM-basierter **Seitenpuffer** (*page cache*) für den NVRAM
  - wobei gepufferte Seiten den zu sichernden **Übergangszustand** erweitern

- der Fall „**NVM-only**“ soll bedeuten, dass die Maschinenprogramme und das Betriebssystem im NVRAM liegend ausgeführt werden
- die damit erreichbare vollständige **Persistenz** aller Programme geht jedoch auf Kosten der **Performanz** der entsprechenden Prozesse
- der Ansatz ist nun, den NVRAM zu virtualisieren und durch DRAM als schnelleren Zwischenspeicher die Zugriffslatenz zu verkürzen
- darüber hinaus ist durch diesen Ansatz eine effiziente Skalierung der Speicherkapazität auf Betriebssystemebene möglich [7]
  - virtueller Speicher ergänzt um NVRAM als schnellen Hintergrundspeicher
  - Hochskalieren von Hauptspeicherkapazität ohne zusätzlichen DRAM

- der Fall „**NVM-only**“ soll bedeuten, dass die Maschinenprogramme und das Betriebssystem im NVRAM liegend ausgeführt werden
- die damit erreichbare vollständige **Persistenz** aller Programme geht jedoch auf Kosten der **Performanz** der entsprechenden Prozesse
- der Ansatz ist nun, den NVRAM zu virtualisieren und durch DRAM als schnelleren Zwischenspeicher die Zugriffslatenz zu verkürzen
- darüber hinaus ist durch diesen Ansatz eine effiziente Skalierung der Speicherkapazität auf Betriebssystemebene möglich [7]



- die im DRAM gepufferten Seiten erweitern den Übergangszustand der Maschinenprogrammprozesse
    - nicht echt, aber echt erscheinend sind die Seiten im DRAM persistent
    - in Wirklichkeit sind diese während ihrer Lagerung im DRAM flüchtig
- ↪ **Konsistenz** der im DRAM gepufferten modifizierten Seiten des NVRAM

- die im DRAM gepufferten Seiten erweitern den Übergangszustand der Maschinenprogrammprozesse
  
  - die Größe dieses Puffers ist durch die Breite des Restenergiefensters bestimmt, nicht durch die Menge an DRAM im System
    - Analyse der Breite des verbleibenden Energiefensters beim Stromausfall
    - Analyse der Größe des zu sichernden volatilen Prozessorzustands
    - Analyse der Dauer der Sicherung eines Datums in den NVRAM
- ↪ kleinste Fensterbreite, größter Zustand und längste Sicherungsdauer

- die im DRAM gepufferten Seiten erweitern den Übergangszustand der Maschinenprogrammprozesse
  
- die Größe dieses Puffers ist durch die Breite des Restenergiefensters bestimmt, nicht durch die Menge an DRAM im System
  
- letztlich ist der Puffer begrenzt durch die **Zeit**, die beim Stromausfall noch verfügbar ist, um Seiten in den NVRAM zurückzuschreiben
  - das Betriebssystem sollte Termine **strikt** einhalten, **echtzeitfähig** sein

- die im DRAM gepufferten Seiten erweitern den Übergangszustand der Maschinenprogrammprozesse
  
  - die Größe dieses Puffers ist durch die Breite des Restenergiefensters bestimmt, nicht durch die Menge an DRAM im System
  
  - letztlich ist der Puffer begrenzt durch die **Zeit**, die beim Stromausfall noch verfügbar ist, um Seiten in den NVRAM zurückzuschreiben
    - das Betriebssystem sollte Termine **strikt** einhalten, **echtzeitfähig** sein
- strikt**
- Terminüberschreitung kann zum Systemversagen führen
  - das Systemversagen kann eine „Katastrophe“ hervorrufen

- die im DRAM gepufferten Seiten erweitern den Übergangszustand der Maschinenprogrammprozesse
  
  - die Größe dieses Puffers ist durch die Breite des Restenergiefensters bestimmt, nicht durch die Menge an DRAM im System
  
  - letztlich ist der Puffer begrenzt durch die **Zeit**, die beim Stromausfall noch verfügbar ist, um Seiten in den NVRAM zurückzuschreiben
    - das Betriebssystem sollte Termine **strikt** einhalten, **echtzeitfähig** sein
- ↪ **eine Terminverletzung der Sicherungsprozedur ist nicht tolerierbar**



- Konsistenz einer Seite im DRAM mit ihrem Original im NVRAM ist sicherzustellen, wann immer sie aus den Puffer entfernt wird

- Konsistenz einer Seite im DRAM mit ihrem Original im NVRAM ist sicherzustellen, wann immer sie aus den Puffer entfernt wird:  
**implizit**
  - als Folge der **Ersetzungsstrategie** (*replacement policy*)
    - wenn die Seite wegen Überbelegung verdrängt werden soll
  - beim **Stromausfall** oder einer anderen **Ausnahmesituation**

- Konsistenz einer Seite im DRAM mit ihrem Original im NVRAM ist sicherzustellen, wann immer sie aus den Puffer entfernt wird:

## **explizit**

- auf Veranlassung des Prozesses selbst, einer **Freigabeoperation**
  - auf der Seite liegen Daten, die zu kommunizieren sind
  - auf der Seite liegen gemeinsame Daten, die zu synchronisieren sind
- das heißt, wenn **kooperierende Prozesse** dieselbe Seite nutzen

- Konsistenz einer Seite im DRAM mit ihrem Original im NVRAM ist sicherzustellen, wann immer sie aus den Puffer entfernt wird:
  - jede **modifizierte Seite** muss in den NVRAM „ausgespült“ werden





Einleitung

Grundlagen

Datenkonsistenz

Ausfalltoleranz

Zugriffslatenz

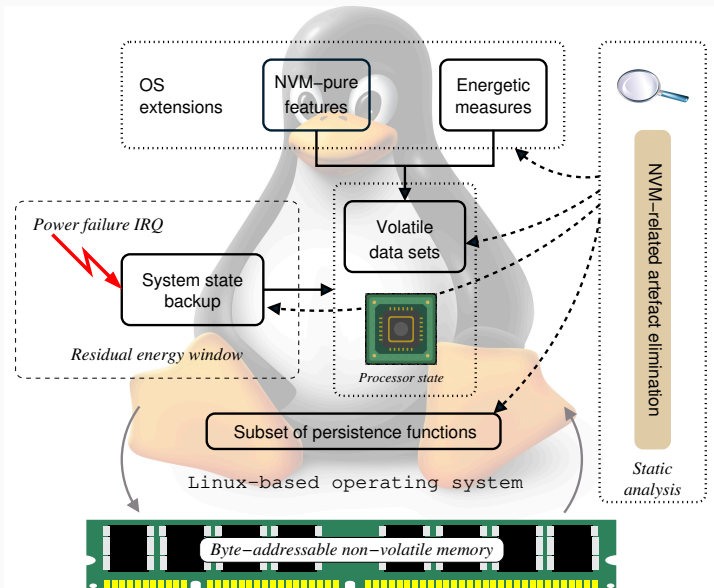
Fallstudien

Rauscharmes System

Kaschierendes System

Resilientes System

Zusammenfassung



*Ein Betriebssystem, das NVM insbesondere auch für eigene Zwecke nutzt, kann auf viele, wenn nicht sämtliche, für gewöhnlich sonst zu realisierende Persistenzmaßnahmen verzichten und dadurch sein Maß an Hintergrundrauschen (background noise) verringern. [6]*

- nichtfunktionale Eigenschaften von Rechensystemen verbessern

*Ein Betriebssystem, das NVM insbesondere auch für eigene Zwecke nutzt, kann auf viele, wenn nicht sämtliche, für gewöhnlich sonst zu realisierende Persistenzmaßnahmen verzichten und dadurch sein Maß an Hintergrundrauschen (background noise) verringern. [6]*

- nichtfunktionale Eigenschaften von Rechensystemen verbessern:
  - BO**
    - energetische Maßnahmen
    - ↪ Energieeffizienzsteigerungen durch „Sekundenschlaf“ mit NVM
    - ↪ dynamische Maßnahmen für Energieeinsparungen zur Laufzeit
    - ↪ Energie betreffende Speicherzugriffskosten

*Ein Betriebssystem, das NVM insbesondere auch für eigene Zwecke nutzt, kann auf viele, wenn nicht sämtliche, für gewöhnlich sonst zu realisierende Persistenzmaßnahmen verzichten und dadurch sein Maß an Hintergrundrauschen (background noise) verringern. [6]*

- nichtfunktionale Eigenschaften von Rechensystemen verbessern:
  - ER
    - fixpunktorientierte Quelltextaufbereitung
    - ↳ Beseitigung der persistenzbezogenen Betriebssystemartefakte
    - ↳ Beseitigung/Begrenzung von Unterbrechungslatenzen
    - ↳ Rekonstruktion sensibler Betriebssystemstrukturen

*Ein Betriebssystem, das NVM insbesondere auch für eigene Zwecke nutzt, kann auf viele, wenn nicht sämtliche, für gewöhnlich sonst zu realisierende Persistenzmaßnahmen verzichten und dadurch sein Maß an Hintergrundrauschen (background noise) verringern. [6]*

- nichtfunktionale Eigenschaften von Rechensystemen verbessern:

### **BO/ER**

- hardware-orientierte Betriebssystempersistenz
  - ↪ Fixpunktmechanismus zur Tolerierung von Stromausfall
  - ↪ Universalbetriebssystem *in situ* NVM
  - ↪ transaktionale nichtblockierende Betriebssystemstrukturen

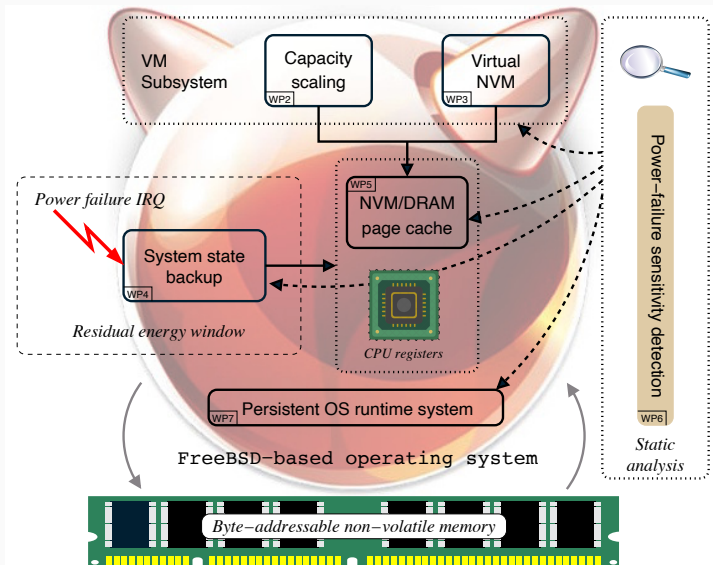
*Ein Betriebssystem, das NVM insbesondere auch für eigene Zwecke nutzt, kann auf viele, wenn nicht sämtliche, für gewöhnlich sonst zu realisierende Persistenzmaßnahmen verzichten und dadurch sein Maß an Hintergrundrauschen (background noise) verringern. [6]*

■ nichtfunktionale Eigenschaften von Rechensystemen verbessern:

**BO**    ■ energetische Maßnahmen

**ER**    ■ fixpunktorientierte Quelltextaufbereitung

**BO/ER**    ■ hardware-orientierte Betriebssystempersistenz



*Ein Betriebssystem, das NVM als integralen Bestandteil des virtuellen Speichers behandelt, ermöglicht die kosteneffiziente Skalierung der Hauptspeicherkapazität, kompensiert die höheren Zugriffszeiten im Vergleich zu herkömmlichen Hauptspeicher und verhindert implizit in Ausnahmefällen potentielle persistente inkonsistente Zustände. [9]*

- Altsoftware unverändert und effizient in NVM ausführen können

*Ein Betriebssystem, das NVM als integralen Bestandteil des virtuellen Speichers behandelt, ermöglicht die kosteneffiziente Skalierung der Hauptspeicherkapazität, kompensiert die höheren Zugriffszeiten im Vergleich zu herkömmlichen Hauptspeicher und verhindert implizit in Ausnahmefällen potentielle persistente inkonsistente Zustände. [9]*

- Altsoftware unverändert und effizient in NVM ausführen können:
  - CB**
    - NVRAM-basierte Skalierung der Hauptspeicherkapazität
    - betriebssystemtaugliches persistentes Laufzeitsystem

*Ein Betriebssystem, das NVM als integralen Bestandteil des virtuellen Speichers behandelt, ermöglicht die kosteneffiziente Skalierung der Hauptspeicherkapazität, kompensiert die höheren Zugriffszeiten im Vergleich zu herkömmlichen Hauptspeicher und verhindert implizit in Ausnahmefällen potentielle persistente inkonsistente Zustände. [9]*

- Altsoftware unverändert und effizient in NVM ausführen können:
  - ER**
    - reaktive Sicherung des flüchtigen Systemzustands
    - Empfindlichkeitserkennung von Software betreffs Stromausfall

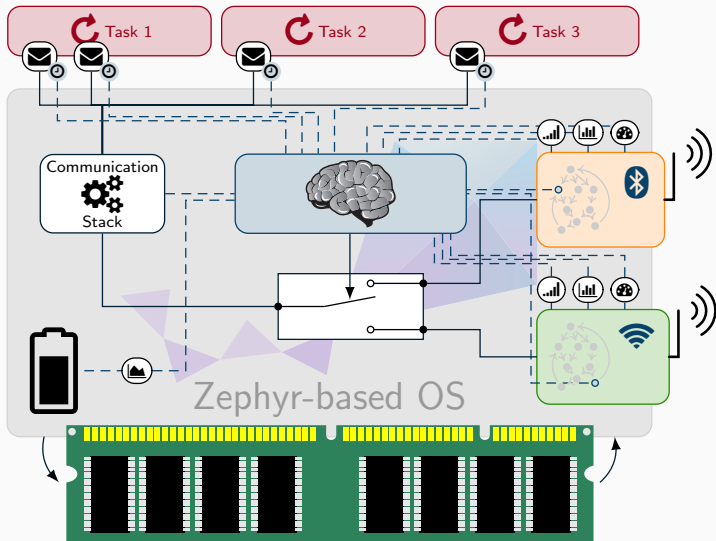
*Ein Betriebssystem, das NVM als integralen Bestandteil des virtuellen Speichers behandelt, ermöglicht die kosteneffiziente Skalierung der Hauptspeicherkapazität, kompensiert die höheren Zugriffszeiten im Vergleich zu herkömmlichen Hauptspeicher und verhindert implizit in Ausnahmefällen potentielle persistente inkonsistente Zustände. [9]*

- Altsoftware unverändert und effizient in NVM ausführen können:

- CB/ER**
  - virtueller nichtflüchtiger Hauptspeicher
  - zum Restenergiefenster konforme Seitenpufferverwaltung

*Ein Betriebssystem, das NVM als integralen Bestandteil des virtuellen Speichers behandelt, ermöglicht die kosteneffiziente Skalierung der Hauptspeicherkapazität, kompensiert die höheren Zugriffszeiten im Vergleich zu herkömmlichen Hauptspeicher und verhindert implizit in Ausnahmefällen potentielle persistente inkonsistente Zustände. [9]*

- Altsoftware unverändert und effizient in NVM ausführen können:
  - CB**
    - NVRAM-basierte Skalierung der Hauptspeicherkapazität
    - betriebssystemtaugliches persistentes Laufzeitsystem
  - ER**
    - reaktive Sicherung des flüchtigen Systemzustands
    - Empfindlichkeitserkennung von Software betreffs Stromausfall
  - CB/ER**
    - virtueller nichtflüchtiger Hauptspeicher
    - zum Restenergiefenster konforme Seitenpufferverwaltung



Transactional Persistence in Non-Volatile Memory

*Ein Betriebs- und Kommunikationssystem, das byteadressierbaren NVM als primären Hauptspeicher begreift und dem Paradigma der transaktionalen Programmierung folgt, macht diese Verbindung zum robusten Bestandteil einer Software, wie sie für „Resilienz in vernetzten Welten“ erforderlich ist. [5]*

- Ausfälle, Überlastung, Angriffe und das Unerwartete meistern

*Ein Betriebs- und Kommunikationssystem, das byteadressierbaren NVM als primären Hauptspeicher begreift und dem Paradigma der transaktionalen Programmierung folgt, macht diese Verbindung zum robusten Bestandteil einer Software, wie sie für „Resilienz in vernetzten Welten“ erforderlich ist. [5]*

- Ausfälle, Überlastung, Angriffe und das Unerwartete meistern:
  - SB**
    - transaktionale Netzwerke
    - NVM-basierte Kommunikationsprotokollstapel
    - Transportresilienz, „*multi-homing*“<sup>5</sup> und „*multi-sourcing*“<sup>6</sup>

---

<sup>5</sup>Ein Gerät verfügt über mehrere Netzwerkadressen.

<sup>6</sup>Netzbetrieb und -infrastruktur leisten mehrere Anbieter.

*Ein Betriebs- und Kommunikationssystem, das bytheadressierbaren NVM als primären Hauptspeicher begreift und dem Paradigma der transaktionalen Programmierung folgt, macht diese Verbindung zum robusten Bestandteil einer Software, wie sie für „Resilienz in vernetzten Welten“ erforderlich ist. [5]*

- Ausfälle, Überlastung, Angriffe und das Unerwartete meistern:

- ER**
    - reaktive Sicherung des flüchtigen Systemzustands
    - NVM-basierter scheinbar nichtflüchtiger Haldenspeicher
    - zeit- und energiebewusste Operationen des Betriebssystemkerns
-

*Ein Betriebs- und Kommunikationssystem, das bytheadressierbaren NVM als primären Hauptspeicher begreift und dem Paradigma der transaktionalen Programmierung folgt, macht diese Verbindung zum robusten Bestandteil einer Software, wie sie für „Resilienz in vernetzten Welten“ erforderlich ist. [5]*

■ Ausfälle, Überlastung, Angriffe und das Unerwartete meistern:

- SB**
  - transaktionale Netzwerke
  - NVM-basierte Kommunikationsprotokollstapel
  - Transportresilienz, „*multi-homing*“<sup>5</sup> und „*multi-sourcing*“<sup>6</sup>
- ER**
  - reaktive Sicherung des flüchtigen Systemzustands
  - NVM-basierter scheinbar nichtflüchtiger Haldenspeicher
  - zeit- und energiebewusste Operationen des Betriebssystemkerns

---

<sup>5</sup>Ein Gerät verfügt über mehrere Netzwerkadressen.

<sup>6</sup>Netzbetrieb und -infrastruktur leisten mehrere Anbieter.

Einleitung

Grundlagen

Datenkonsistenz

Ausfalltoleranz

Zugriffslatenz

Fallstudien

Rauscharmes System

Kaschierendes System

Resilientes System

Zusammenfassung



- **NVRAM** — Hauptspeicher, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert
  - byteadressierbarer nichtflüchtiger Speicher (*non-volatile memory*, NVM)
  - Ersatz von oder Ergänzung zu DRAM, aber auch **disruptive Technologie**

- **NVRAM** — Hauptspeicher, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert
- **Stromausfälle** können dann Abläufe bewirken, die einen sequentiellen Prozess wie einen nichtsequentiellen Prozess erscheinen lassen
  - nämlich wenn das Programm direkt im NVRAM zur Ausführung kommt
  - unerwartet inkonsistente nichtflüchtig gespeicherte Daten sind möglich

- **NVRAM** — Hauptspeicher, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert
- **Stromausfälle** können dann Abläufe bewirken, die einen sequentiellen Prozess wie einen nichtsequentiellen Prozess erscheinen lassen
- in dem jeweils gegebenen **Restenergiefenster** der Stromversorgung den **Übergangszustand** der CPU in den NVM sichern
  - ausgelöst durch einen „*power failure interrupt*“ (PFI)
  - wobei keine **Unterbrechungssperre** diese Maßnahme verhindern darf

- **NVRAM** — Hauptspeicher, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert
- **Stromausfälle** können dann Abläufe bewirken, die einen sequentiellen Prozess wie einen nichtsequentiellen Prozess erscheinen lassen
- in dem jeweils gegebenen **Restenergiefenster** der Stromversorgung den **Übergangszustand** der CPU in den NVM sichern
- DRAM würde nur benötigt, um die bei NVRAM noch vorhandenen höheren **Zugriffszeiten/Latenzen** zu kaschieren
  - Skalierung der Hauptspeicherkapazität durch Virtualisierung des NVRAM
  - DRAM-basierter Seitenpuffer für NVRAM  $\rightsquigarrow$  Übergangszustand des BS

- **NVRAM** — Hauptspeicher, der die gespeicherte Information auch nach Abschalten der Versorgungsspannung nicht verliert
- **Stromausfälle** können dann Abläufe bewirken, die einen sequentiellen Prozess wie einen nichtsequentiellen Prozess erscheinen lassen
- in dem jeweils gegebenen **Restenergiefenster** der Stromversorgung den **Übergangszustand** der CPU in den NVM sichern
- DRAM würde nur benötigt, um die bei NVRAM noch vorhandenen höheren **Zugriffszeiten/Latenzen** zu kaschieren

- [1] ACTIVE POWER, INC.:  
**15 Seconds versus 15 Minutes: Designing for High Availability.**  
2007 (107). –  
White Paper
- [2] BOWER, J. L. ; CHRISTENSEN, C. M.:  
**Disruptive Technologies: Catching the Wave.**  
In: *Harvard Business Review* 73 (1995), Jan.-Febr., Nr. 1, S. 43–53
- [3] DANNEELS, E. :  
**Disruptive Technology Reconsidered: A Critique and Research Agenda.**  
In: *The Journal of Product Innovation Management* 21 (2004), Jul., Nr. 4, S. 246–258
- [4] DENNING, P. J.:  
**The Working Set Model for Program Behavior.**  
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 323–333

- [5] HERFET, T. ; SCHRÖDER-PREIKSCHAT, W. :  
**Resilient Power-Constrained Embedded Communication Terminals (ResPECT) / Deutsche Forschungsgemeinschaft (DFG).**  
2022 (TBA (HE 2584/7 and SCHR 603/19)). –  
Research Grant within Priority Programme 2378
- [6] HÖNIG, T. ; SCHRÖDER-PREIKSCHAT, W. :  
**Nichtflüchtigkeit in energiebewussten Betriebssystemen (NEON) / Deutsche Forschungsgemeinschaft (DFG).**  
2021 (465958100 (HO 6277/1 and SCHR 603/16)). –  
Einzelförderung
- [7] KANNAN, S. ; GAVRILOVSKA, A. ; SCHWAN, K. :  
**pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage.**  
In: *Proceedings of the Eleventh European Conference on Computer Systems.*  
New York, NY, USA : Association for Computing Machinery, 2016 (EuroSys '16). –  
ISBN 9781450342407, S. 13:1–13:16

- [8] NARAYANAN, D. ; HODSON, O. :  
**Whole-system persistence.**  
In: *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*, 2012, S. 401–410
- [9] NOLTE, J. ; SCHRÖDER-PREIKSCHAT, W. :  
**Power-Fail Aware Byte-Addressable Virtual Non-Volatile Memory (PAVE) / Deutsche Forschungsgemeinschaft (DFG).**  
2022 (501993201 (NO 625/15 and SCHR 603/18)). –  
Research Grant within Priority Programme 2377
- [10] PENG, I. B. ; GOKHALE, M. B. ; GREEN, E. W.:  
**System Evaluation of the Intel Optane Byte-Addressable NVM.**  
In: *Proceedings of the International Symposium on Memory Systems.*  
New York, NY, USA : Association for Computing Machinery, 2019 (MEMSYS '19). –  
ISBN 9781450372060, S. 304–315

[11] RANSFORD, B. ; LUCIA, B. :

**Nonvolatile Memory is a Broken Time Machine.**

In: *Proceedings of the Workshop on Memory Systems Performance and Correctness.*

New York, NY, USA : Association for Computing Machinery, 2014 (MSPC '14). – ISBN 9781450329170, S. 5:1–5:3

[12] SCHRÖDER-PREIKSCHAT, W. :

**Virtuell gemeinsamer Speicher.**

In: LEHRSTUHL INFORMATIK 4 (Hrsg.): *Betriebssystemtechnik – Adressräume: Trennung, Zugriff, Schutz.*

FAU Erlangen-Nürnberg, 2022 (Vorlesungsfolien), Kapitel 11

[13] UHLIG, V. ; LEVASSEUR, J. ; SKOGLUND, E. ; DANNOWSKI, U. :

**Towards Scalable Multiprocessor Virtual Machines.**

In: *Proceedings of the 3rd Conference on Virtual Machine Research And Technology Symposium.*

USA : USENIX Association, 2004 (VM'04), S. 4:1–4:14