Weak Persistency Semantics from the Ground Up:
Formalising the Persistency Semantics of ARMv8 & Transactional Models

Azalea Raad       John Wickerson       Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)
Imperial College London

azalea@mpi-sws.org    SoundAndComplete.org    @azalearaad
Computer Storage

✓ fast
✗ volatile

✓ fast
✗ volatile

✗ slow
✓ persistent
What is Non-Volatile Memory (NVM)?

*NVM: Hybrid Storage + Memory*

Best of both worlds:

✓ *persistent* (like HDD)
✓ *fast, random access* (like RAM)
Q: Why *Formal* NVM Semantics?

Volatile memory

```plaintext
// x = 0
x := 1
// x = 1
// no recovery
// x = 0
```
Q: Why *Formal* NVM Semantics?

**Volatile** memory

```plaintext
// x = 0
x := 1
// x = 1
// no recovery
// x = 0
```

**Non-Volatile** memory

```plaintext
// x = 0
x := 1
// x = 1
// recovery routine
// x = 1
```
Q: Why *Formal* NVM Semantics?

A: Program *Verification*

Volatile memory

```plaintext
// x = 0
x := 1
// x = 1
// no recovery
// x = 0
```

Non-Volatile memory

```plaintext
// x = 0
x := 1
// x = 1
// recovery routine
// x = 1
```
Q: Why *Formal* NVM Semantics?

What about *Concurrency*?

```plaintext
// x = y = ... = 0
C_1 \parallel C_2 \parallel \ldots \parallel C_n
  // ???

// recovery routine
  // ???
```
Formal Semantic Models

![Diagram showing the evolution of formal semantic models with time and difficulty dimensions.](#)

- Sequential (1940s)
- SC (1979)
Formal Semantic Models

Difficulty

Sequential (1940s)  SC (1979)  WMC (1990s)

(1940s)  (1979)  (1990s)

time
Weak Memory Consistency (WMC)

No total execution order \((\text{to})\) \(\Rightarrow\)

\textit{weak} behaviour absent under SC, caused by:

- instruction \textit{reordering} by compiler
- write propagation across \textit{cache hierarchy}
Weak Memory Consistency (WMC)

No total execution order (to) ⇒

weak behaviour absent under SC, caused by:

- instruction reordering by compiler
- write propagation across cache hierarchy

**Consistency Model**

the *order* in which writes are made visible to other threads

e.g. TSO, ARMv8, POWER, C11, Java
Formal Semantic Models

Difficulty

Sequential (1940s)  SC (1979)  WMC (1990s)  WNVMC (2017)

This Talk
What Can Go Wrong?

// x=y=0
x := 1;
y := 1;

// recovery routine

// x=y=1  OR  x=y=0  OR  x=1; y=0  OR  x=0; y=1
What Can Go Wrong?

// x=y=0
x := 1;
y := 1;

// recovery routine

// x=y=1  OR  x=y=0  OR  x=1;y=0  OR  x=0;y=1

!! Execution continues *ahead of persistence*
— *asynchronous* persists
What Can Go Wrong?

// x=y=0
x := 1;
y := 1;

// recovery routine

// x=y=1 OR x=y=0 OR x=1;y=0 OR x=0;y=1

!! Execution continues *ahead of persistence*
- *asynchronous* persists

!! Writes may persist *out of order*
- *relaxed* persists
What Can Go Wrong?

**Consistency Model**

the order in which writes are made visible to other threads

**Persistency Model**

the order in which writes are persisted to NVM
What Can Go Wrong?

**Consistency Model**

the order in which writes are made visible to other threads

**Persistency Model**

the order in which writes are persisted to NVM

**NVM Semantics**

Consistency + Persistency Model
This Talk

**PARMv8**

(Persistent ARMv8):

*NVM Semantics*

of the

**ARMv8 Architecture**
Challenge #1: *Relaxed* Persists

```
// x=0; y=0
x := 1;
y := 1;

// recovery routine
// x=1; y=1  OR  x=0; y=0  OR  x=1; y=0  OR  x=0; y=1
```

!! *out of order persists*
Persist Barriers: Desiderata

// x=0; y=0

x := 1;

y := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

!! out of order persists

慎重 persist barriers?
Persist Barriers: **Desiderata**

// x=0; y=0

// x=1; y=1

!! out of order persists

⇒ persist barriers?

ARMv8 *does not provide* persist barriers!

ARMv8 memory barriers (e.g. DSB-full) *do not enforce persist* ordering!
Challenge #2: **Asynchronous** Persists

// x=0; y=0
x := 1;
y := 1;

// recovery routine

// x=1; y=1 OR [x=0; y=0] OR x=1; y=0 OR x=0; y=1

!! Execution continues *ahead of persistence*
Explicit Persists: **Desiderata**

```plaintext
// x=0; y=0
x := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

!! Execution continues **ahead of persistence**

**explicit persists?**
Explicit Persists: *Reality on ARMv8*

```c
// x=0; y=0
x := 1;
DC-CVAP x;
y := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

!! Execution continues *ahead of persistence*

lehem persist?
Explicit Persists: *Reality on ARMv8*

```
// x=0; y=0
x := 1;
```

```
DC-CVAP x;
y := 1;
```

// recovery routine

```
// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1
```

 Execution continues *ahead of persistence*

```
DC-CVAP x: asynchronously persist cache line containing x
```

Explicit persists?
Explicit Persists: *Reality on ARMv8*

```c
// x=0; y=0
x := 1;
DC-CVAP x;

// x=1; y=0
x=0; y=1

!! Execution continues *ahead of persistence*

*explicit persists?*

DC-CVAP x: *asynchronously* persist cache line containing x
```

ARMv8 explicit persists are themselves *asynchronous*!
Solution: *Persist Sequence*

```plaintext
// x=0; y=0

x := 1;

DC-CVAP x;
DSB-full;

y := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1
```
Solution: *Persist Sequence*

// x=0; y=0

x := 1;

DC-CVAP x;
DSB-full;

y := 1;

// recovery routine

// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1

⚠️ *Waits* until earlier writes on x are persisted

✓ *synchronous* persists
Solution: *Persist Sequence*

```plaintext
// x=0; y=0
x := 1;
DC-CVAP x;
DSB-full;
y := 1;

// recovery routine
// x=1; y=1 OR x=0; y=0 OR x=1; y=0 OR x=0; y=1
```

- **Waits** until earlier writes on `x` are persisted
- **Disallows reordering**
- ✓ *synchronous* persists
- ✓ *no out of order* persists
“a DSB-full will not complete until all previous DC-CVAP have completed”

“DC-CVAP executes in program order relative to writes to an address in the same cache line”
“a **DSB-full** will not complete until all previous **DC-CVAP** have completed”

“**DC-CVAP** executes in program order relative to writes to an address in the same cache line”

**Ambiguities in text!**
"a DSB-full will not complete until all previous DC-CVAP have completed"

"DC-CVAP executes in program order relative to writes to an address in the same cache line"

Ambiguities in text!
Problem

ambiguous text
counter-intuitive semantics
low-level hardware details

Solution

high-level, hardware-agnostic
NVM libraries:

Persistent Transactions

Ambiguities in text!
What is a Transaction?

Concurrency control mechanism:

- **atomic** work unit:
  - all-or-nothing writes

- **consistent** (e.g. serialisable)

\[
T:\[
\begin{align*}
x &:= 1; \\
y &:= 1;
\end{align*}
\]

// \(x = y = 0\) OR \(x = y = 1\)
What is a **Persistent** Transaction?

Concurrency & **persistency** control mechanism:

- **atomic** work unit:
  - all-or-nothing writes
  - all-or-nothing **persists**

- **consistent** (e.g. serialisable)

```plaintext
T: [x := 1;
y := 1;]

// recovery routine
// x = y = 0 OR x = y = 1
```

// x = y = 0

// x = y = 0 OR x = y = 1
What is a *Persistent* Transaction?

Concurrency & *persistency* control mechanism:

- **atomic** work unit:
  - all-or-nothing writes
  - all-or-nothing *persists*

- **consistent** (e.g. serialisable)

- **persistent** (e.g. *persistently serialisable*):

```
T: [x := 1;
  y := 1;]

// x = y = 0
// recovery routine
// x = y = 0   OR   x = y = 1
```
Serialisability (SER)

*All* transactions appear to **execute** in a sequential order.
**Persistent Serialisability (PSER)**

*All* transactions appear to **execute** in a sequential order.

*A prefix* of transactions appears to **persist** in the **same sequential order**.

\[ T_1 : \begin{cases} x := 1; \\ a := y; \end{cases} \quad \begin{cases} y := 1; \\ b := x; \end{cases} T_2 \]

\[ T_1 \rightarrow T_2 \quad T_2 \rightarrow T_1 \]
**Persistent Serialisability (PSER)**

*All* transactions appear to **execute** in a sequential order.

*A prefix* of transactions appears to **persist** in the **same sequential order**.

\[
\begin{align*}
T1: & \quad \begin{cases} x := 1; \\ a := y; \end{cases} \\
T2: & \quad \begin{cases} y := 1; \\ b := x; \end{cases}
\end{align*}
\]

// \( x = y = 0 \)
**Persistent Serialisability (PSER)**

*All* transactions appear to **execute** in a sequential order.

A **prefix** of transactions appears to **persist** in the **same sequential order**.
**Persistent Serialisability (PSER)**

All transactions appear to **execute** in a sequential order.

A **prefix** of transactions appears to **persist** in the **same sequential order**.

```
T1: [x := 1; a := y;]
T2: [y := 1; b := x;]
```

- T1 → T2
- T2 → T1

// x = y = 0

// x = 1, y = 0

// x = y = 1
**Persistent Serialisability (PSER)**

*All* transactions appear to **execute** in a sequential order.

*A prefix* of transactions appears to **persist** in the *same sequential order*.

\[ \text{T1} \rightarrow \text{T2} \rightarrow \text{T3} \rightarrow \text{T4} \rightarrow \text{T5} \rightarrow \text{T6} \rightarrow \text{T7} \rightarrow \ldots \]

*all persist*  \hspace{2cm} *none persist*
**Persistent Serialisability (PSER)**

*All* transactions appear to **execute** in a sequential order.

A **prefix** of transactions appears to **persist** in the **same sequential order** in **each era**.

---

![Diagram illustrating Persistent Serialisability (PSER)](image)

---

- **execution**
- **recovery**
- **no crashes**
**Persistent Serialisability (PSER)**

All transactions appear to **execute** in a sequential order.

A prefix of transactions appears to **persist** in the same sequential order in **each era**.

**PSER**

- **Strong guarantees**
- **Intuitive semantics**

No crashes
Persistent Serialisability (PSER)

All transactions appear to execute in a sequential order.

A prefix of transactions appears to persist in the same sequential order in each era.

PSER Evaluation

1. Is PSER feasible?
2. Is PSER useful?
Is PSER *Feasible*?

✓ **PSER implementation in ARM**

Take **SER Implementation** — e.g. 2-PL

- add code for *persistence* — i.e. persist sequences
- add code to *log metadata* for *recovery*
- add *recovery mechanism*
Is PSER **Feasible**?

✓ PSER *implementation* in ARM

Take **SER** Implementation — e.g. 2-PL

- add code for *persistence* — i.e. persist sequences
- add code to *log metadata* for *recovery*
- add *recovery mechanism*

**recovery mechanism**

check log for *incomplete* transactions:

either *complete*

or *rollback*
Is PSER **Feasible**?

✓ PSER *implementation* in ARM

Take SER Implementation — e.g. 2-PL

✦ add code for *persistence* — i.e. persist sequences
✦ add code to *log metadata* for recovery

---

**Yes!**

Correct Implementation in PARMv8

⇒ check log for *incomplete* transactions:

  either *complete*
  or *rollback*
Is PSER *Useful*?

Given library $L$ (e.g. queue library):

1. Take *any* correct *sequential* implementation of $L$

```
enq(q,v) =
  <enq_body>

deq(q) =
  <deq_body>
```

sequential queue imp.
Is PSER *Useful*?

Given library $L$ (e.g. queue library):

1. Take *any* correct *sequential* implementation of $L$
2. wrap each operation in a PSER transaction

---

Sequential queue imp.

```plaintext
\[
\begin{align*}
\text{enq} (q, v) &= <\text{enq}_\text{body}> \\
\text{deq} (q) &= <\text{deq}_\text{body}>
\end{align*}
\]

```
Is PSER **Useful**?

Given library $L$ (e.g. queue library):

1. Take *any* correct **sequential** implementation of $L$
2. wrap each operation in a PSER transaction

$\implies$ **correct, concurrent & persistent** implementation of $L$

**enq (q, v) =**

<enq_body>

**deq (q) =**

<deq_body>

sequential queue imp.

**enq (q, v) =**

pser{
<enq_body>
}

**deq (q) =**

pser{
<deq_body>
}

**correct concurrent & persistent** queue imp.
Is PSER **Useful**?

Given library $L$ (e.g. queue library):

1. Take any correct sequential implementation of $L$

2. wrap each operation in a PSER transaction

$\Rightarrow$ any correct sequential implementation $\Rightarrow$ correct, concurrent & persistent implementation

sequential queue imp.

correct concurrent & persistent queue imp.
Summary

✓ Formalised **architecture-level** NVM semantics:
  ✦ **PARMv8**

✓ Formalised **language-level** NVM semantics:
  ✦ **PSER**

✓ More in the paper
  ✦ **General framework** for declarative persistency

❓ Future Work:
  ✦ program logics
  ✦ model checking algorithms
Summary

✓ Formalised **architecture-level** NVM semantics:
  ✧ **PARMv8**

✓ Formalised **language-level** NVM semantics:
  ✧ **PSER**

✓ More in the paper
  ✧ **General framework** for declarative persistency

❓ Future Work:
  ✧ program logics
  ✧ model checking algorithms

Thank You for Listening!