Zen and the Art of Concurrency Control
An Exploration of TM Safety Property Space with Early Release in Mind

Konrad Siek and Paweł T. Wojciechowski
Poznań University of Technology
{konrad.siek,pawel.t.wojciechowski}@cs.put.edu.pl

14 VII 2014

http://dsg.cs.put.poznan.pl
Quality is not absolutely applicable → depends on the situation

quality = TM safety
situation = high contention
Quality is not absolutely applicable
→ depends on the situation

[source: wikipedia]
Quality is not absolutely applicable
→ depends on the situation

quality = TM safety
situation = high contention
High contention

\[ T_1 \left[ r(x)0, w(x)1 \right] \]
\[ T_2 \left[ r(x)0, w(x)1 \cup r(x)1, w(x)2 \right] \]
\[ T_3 \left[ r(x)0, w(x)1 \cup r(x)1, w(x)2 \cup r(x)2, w(x)3 \right] \]
\[ T_4 \left[ r(x)0, w(x)1 \cup r(x)1, w(x)2 \cup r(x)2, w(x)3 \cup r(x)3, w(x)4 \right] \]
High contention

\[ T_1 \left[ r(x)0, w(x)1 \right] \]
\[ T_2 \left[ r(x)0, w(x)1 \Leftrightarrow \left[ r(x)1, w(x)2 \right] \right] \]
\[ T_3 \left[ r(x)0, w(x)1 \Leftrightarrow \left[ r(x)1, w(x)2 \Leftrightarrow \left[ r(x)2, w(x)3 \right] \right] \right] \]
\[ T_4 \left[ r(x)0, w(x)1 \Leftrightarrow \left[ r(x)1, w(x)2 \Leftrightarrow \left[ r(x)2, w(x)3 \Leftrightarrow \left[ r(x)3, w(x)4 \right] \right] \right] \right] \]
Early release

\[ T_1 \left[ r(x)0, w(x)1, r(y)0, w(y)1 \right] \]
\[ T_2 \quad \xrightarrow{\quad} \quad \left[ r(x)1, w(x)2, r(y)1, w(y)2 \right] \]
Early release

\[
T_1 \left[ \ r(x)0, \ w(x)1, \ r(y)0, \ w(y)1 \ \right]
\]
\[
T_2 \left[ \ r(x)1, \ w(x)2, \ r(y)1, \ w(y)2 \ \right]
\]
Early release

\[ T_1 \left[ r(x)0, w(x)1, r(y)0, w(y)1 \right] \]

\[ T_2 \left[ r(x)1, w(x)2, r(y)1, w(y)2 \right] \]
\[
T_1 \left[ r(x)0, w(x)1, r(y)0, w(y)1 \right] \\
T_2 \left[ \neg r(x)1, w(x)2, \neg r(y)1, w(y)2 \right]
\]

Which TM safety properties can be used for early release?
Early release

**Definition**

Transaction $T_i$ releases $x$ early in $H$ iff there is some prefix $H'$ of $H$, such that $T_i$ is live in $H'$ and there exists $T_j$ in $H'$ such that there is a non-local read operation $op_j$ in $H' | T_j$ reading $v$ from $x$ and a preceding write operation $op_i$ in $H' | T_i$ writing $x$ to $v$.

Example:

\[
T_1 \llbracket r(x)0, w(x)1, r(y)0, w(y)1 \rrbracket \\
T_2 \llbracket \swarrow r(x)1, w(x)2, \swarrow r(y)1, w(y)2 \rrbracket
\]
Serializability

**Definition**

History $H$ is serializable iff there exists some linear extension (sequential witness history) $\hat{S}$ such that $\hat{S}$ only contains legal transactions.
**Definition**

History $H$ is serializable iff there exists some linear extension (sequential witness history) $\hat{S}$ such that $\hat{S}$ only contains legal transactions.

Example:

$T_1 \left[ r(x)0, w(x)1, r(y)0, w(y)1 \right]$

$T_2 \left[ \rightarrow r(x)1, w(x)2, \rightarrow r(y)1, w(y)2 \right]$
Definition

History $H$ is serializable iff there exists some linear extension (sequential witness history) $\hat{S}$ such that $\hat{S}$ only contains legal transactions.

Example:

$T_1 \left[ \begin{array}{c} r(x)0, w(x)1, r(y)0, w(y)1 \end{array} \right]$

$T_2 \left[ \begin{array}{c} \longleftarrow r(x)1, w(x)2, \longleftarrow r(y)1, w(y)2 \end{array} \right]$

$\hat{S} = \langle T_1, T_2 \rangle$
**Serializability**

**Definition**

History $H$ is serializable iff there exists some linear extension (sequential witness history) $\hat{S}$ such that $\hat{S}$ only contains legal transactions.

Example:

$$T_1 \left[ \begin{array}{c} r(x)0, \ w(x)1, \ r(y)0, \ w(y)1 \end{array} \right]$$

$$T_2 \left[ \begin{array}{c} \not{r(x)}1, \ w(x)2, \ \not{r(y)}1, \ w(y)2 \end{array} \right]$$

$$\hat{S} = \langle T_1, T_2 \rangle$$

A serializable history can contain early release.
Components of opacity:

- Serializability
- Real-time order
- Consistency
Components of opacity:

- Serializability
- Real-time order
- Consistency

**Definition**

Non-local $op_r$ in $T_i$ ($i \neq 0$) is consistent if there is a preceding non-local write operation writing $v$ to $x$ in $H|T_k$ ($T_k \neq T_i$) where $T_k$ is committed or commit-pending.
Opacity

Components of opacity:

- Serializability
- Real-time order
- Consistency

Definition

Non-local \( op_r \) in \( T_i \) \((i \neq 0)\) is consistent if there is a preceding non-local write operation writing \( v \) to \( x \) in \( H|T_k \ (T_k \neq T_i) \) where \( T_k \) is committed or commit-pending.
Opacity

Components of opacity:
- Serializability
- Real-time order
- Consistency

Definition

Non-local $op_r$ in $T_i$ ($i \neq 0$) is consistent if there is a preceding non-local write operation writing $v$ to $x$ in $H|T_k$ ($T_k \neq T_i$) where $T_k$ is committed or commit-pending.

Live transaction $\neq$ committed or commit-pending.
Opacity

Components of opacity:
- Serializability
- Real-time order
- Consistency

Definition

Non-local \( op_r \) in \( T_i \) \((i \neq 0)\) is consistent if there is a preceding non-local write operation writing \( v \) to \( x \) in \( H|T_k \) \((T_k \neq T_i)\) where \( T_k \) is committed or commit-pending.

Live transaction \( \neq \) committed or commit-pending.

An opaque history cannot contain early release.
Definition

History $H$ is elastic opaque iff there exists a cutting function $f_C$ that replaces each elastic transaction $T_i$ in $H$ with its consistent well-formed cut $C_t$, such that $f_C(H)$ is opaque.
Definition

History $H$ is elastic opaque iff there exists a cutting function $f_C$ that replaces each elastic transaction $T_i$ in $H$ with its consistent well-formed cut $C_t$, such that $f_C(H)$ is opaque.

Example:

$$
\begin{align*}
T_1 & \left[ r(y)0, w(x)1, r(x)1, r(y)0 \right] \\
T_2 & \left[ \overset{\longrightarrow}{r(x)1} \right]
\end{align*}
$$

$\left\{ \begin{array}{c}
T_1 \\
T_2
\end{array} \right\} H$

**Elastic opacity**

**Definition**

History $H$ is elastic opaque iff there exists a cutting function $f_C$ that replaces each elastic transaction $T_i$ in $H$ with its consistent well-formed cut $C_t$, such that $f_C(H)$ is opaque.

Example:

$$
\begin{align*}
T_1 & \left[ \quad r(y)0, \quad w(x)1, \quad r(x)1, \quad r(y)0 \quad \right] \\
T_2 & \left[ \quad \quad \rightsquigarrow r(x)1 \quad \right] \\
\end{align*}
\} \quad H
$$

$$
\begin{align*}
T_1' & \left[ \quad r(y)0, \quad w(x)1 \quad \right] \\
T_2 & \left[ \quad \quad \rightsquigarrow r(x)1 \quad \right] \\
T_1'' & \left[ \quad r(x)1, \quad r(y)0 \quad \right] \\
\} \quad f_C(H)
\end{align*}
$$
Elastic opacity

**Definition**

History $H$ is elastic opaque iff there exists a cutting function $f_C$ that replaces each elastic transaction $T_i$ in $H$ with its consistent well-formed cut $C_t$, such that $f_C(H)$ is opaque.

Example:

$T_1$ \[
\begin{array}{l}
\{ r(y)0, w(x)1, \\
\quad r(x)1, r(y)0 \\
\} \\
\end{array}
\]

$T_2$ \[
\begin{array}{l}
\{ \nrightarrow r(x)1 \\
\} \\
\end{array}
\]

$\{ T_1, T_2 \} \} H$

$T'_1$ \[
\begin{array}{l}
\{ r(y)0, w(x)1 \} \\
\end{array}
\]

$T_2$ \[
\begin{array}{l}
\{ \nrightarrow r(x)1 \} \\
\end{array}
\]

$T''_1$ \[
\begin{array}{l}
\{ r(x)1, r(y)0 \} \\
\end{array}
\]

$\{ T'_1, T_2, T''_1 \}$ \} $f_C(H)$

An elastic opaque history can contain early release.
Elastic opacity

**Definition**

History $H$ is elastic opaque iff there exists a cutting function $f_C$ that replaces each elastic transaction $T_i$ in $H$ with its consistent well-formed cut $C_t$, such that $f_C(H)$ is opaque.

Example:

$$
\begin{align*}
T_1 & \left[ \ r(y)0, w(x)1, \quad r(x)1, r(y)0 \ \right] \\
T_2 & \left[ \ \overset{\rightarrow}{r(x)1} \ \right] \\
& \left\{ \right. \\
T_1' & \left[ \ r(y)0, w(x)1 \ \right] \\
T_2 & \left[ \ \overset{\rightarrow}{r(x)1} \ \right] \\
T_1'' & \left[ \ r(x)1, r(y)0 \ \right] \\
& \left\{ \right. \\
& \left. \right\} \quad f_C(H)
\end{align*}
$$

An elastic opaque history can contain early release. However...
Elastic opacity

Well-formed cut:

- A subhistory cannot start with a write (unless it is the first subhistory of a cut).
- If there are two writes in a transaction, they are within the same subhistory.
- A subhistory cannot be shorter than two operations (unless the transaction contains only one operation).
Elastic opacity

Well-formed cut:

- A subhistory cannot start with a write (unless it is the first subhistory of a cut).
- If there are two writes in a transaction, they are within the same subhistory.
- A subhistory cannot be shorter than two operations (unless the transaction contains only one operation).

\[
\begin{align*}
T_1 & \left[ r(y)0, w(x)1, r(x)1, r(y)0 \right] \\
T_2 & \left[ r(x)1 \right] \\
\end{align*}
\]  \hspace{1cm} \left\{ \begin{array}{c}
H \\
\end{array} \right. \\

\[
\begin{align*}
T_1' & \left[ r(y)0, w(x)1 \right] \\
T_2 & \left[ r(x)1 \right] \\
T_1'' & \left[ r(x)1, r(y)0 \right] \\
\end{align*}
\]  \hspace{1cm} \left\{ \begin{array}{c}
f_C(H) \\
\end{array} \right. \\

Elastic opacity

Well-formed cut:

- A subhistory cannot start with a write (unless it is the first subhistory of a cut).
- If there are two writes in a transaction, they are within the same subhistory.
- A subhistory cannot be shorter than two operations (unless the transaction contains only one operation).

\[
T_1 \ [ r(y)0, w(x)1, \ r(x)1 ] \\
T_2 \ [ \overset{\rightarrow}{r(x)}1 ] \ \\
\{ \overset{\rightarrow}{H} \}
\]

\[
T'_1 \ [ r(y)0, w(x)1 ] \\
T_2 \ [ \overset{\rightarrow}{r(x)}1 ] \\
T''_1 \ [ \overset{\rightarrow}{r(x)}1 ] \ \\
\{ f_C(H) \}
\]
Definition (TMS1—Valid Response)

For operation $op$ to return in some subhistory $H|T_i$, there must exist some set of transactions $S$ that follow real-time order and justify the legality of $op$, and for any $T_j \in S$ it is true that,

- if $T_j$ precedes $T_i$ in real-time order then $T_j$ is committed, or
- $T_j$ is committed or commit-pending otherwise.
Definition (TMS1—Valid Response)

For operation \( op \) to return in some subhistory \( H|T_i \), there must exist some set of transactions \( S \) that follow real-time order and justify the legality of \( op \), and for any \( T_j \in S \) it is true that,

- if \( T_j \) precedes \( T_i \) in real-time order then \( T_j \) is **committed**, or
- \( T_j \) is **committed or commit-pending** otherwise.
Definition (TMS1—Valid Response)

For operation $op$ to return in some subhistory $H|T_i$, there must exist some set of transactions $S$ that follow real-time order and justify the legality of $op$, and for any $T_j \in S$ it is true that,

- if $T_j$ precedes $T_i$ in real-time order then $T_j$ is committed, or
- $T_j$ is committed or commit-pending otherwise.

Live transaction $\neq$ committed or commit-pending.
Definition (TMS1—Valid Response)

For operation \( op \) to return in some subhistory \( H|T_i \), there must exist some set of transactions \( S \) that follow real-time order and justify the legality of \( op \), and for any \( T_j \in S \) it is true that,

- if \( T_j \) precedes \( T_i \) in real-time order then \( T_j \) is committed, or
- \( T_j \) is committed or commit-pending otherwise.

Live transaction \( \neq \) committed or commit-pending.

A TMS1 history cannot contain early release.
TMS1 & TMS2

Definition (TMS1—Valid Response)

For operation $op$ to return in some subhistory $H|T_i$, there must exist some set of transactions $S$ that follow real-time order and justify the legality of $op$, and for any $T_j \in S$ it is true that,

- if $T_j$ precedes $T_i$ in real-time order then $T_j$ is committed, or
- $T_j$ is committed or commit-pending otherwise.

Live transaction $\neq$ committed or commit-pending.

A TMS1 history cannot contain early release.

A TMS2 history cannot contain early release (TMS2 $\subset$ TMS1).
Virtual world consistency

**Definition**

History $H$ is VWC iff all committed transactions are strict serializable, and for all aborted transactions there exists a linear extension of its causal past that is legal.
Virtual world consistency

Definition

History $H$ is VWC iff all committed transactions are \textbf{strict serializable}, and for all aborted transactions there exists a linear extension of its causal past that is legal.
Virtual world constistency

**Definition**

History $H$ is VWC iff all committed transactions are **strict serializable**, and for all aborted transactions there exists a linear extension of its causal past that is legal.

Example:

\[
T_i \left[ r(x)0, w(x)1, r(y)0 \right] \\
T_j \left[ \rightsquigarrow r(x)1 \right]
\]
Virtual world consistency

**Definition**

History $H$ is VWC iff all committed transactions are **strict serializable**, and for all aborted transactions there exists a linear extension of its causal past that is legal.

**Example:**

$$
T_i \left[ r(x)0, w(x)1, r(y)0 \right]
$$

$$
T_j \left[ \leftarrow r(x)1 \right]
$$

$$
\hat{S} = \langle T_1, T_2 \rangle
$$
Virtual world consistency

**Definition**

History $H$ is VWC iff all committed transactions are **strict serializable**, and for all aborted transactions there exists a linear extension of its causal past that is legal.

Example:

$$T_i \ [ r(x)0, w(x)1, r(y)0 ]$$

$$T_j \ [ \leftarrow r(x)1 ]$$

$$\hat{S} = \langle T_1, T_2 \rangle$$

A VWC history can contain early release.
Virtual world consistency

If $T_i$ releases early in $H$, then $T_i$ cannot abort.
Virtual world consistency

If $T_i$ releases early in $H$, then $T_i$ cannot abort.

$T_i [ r(x)0, w(x)1, r(y)0 \Leftarrow$

$T_j [ \Rightarrow r(x)1 ]$
Virtual world consistency

If $T_i$ releases early in $H$, then $T_i$ cannot abort.

$$T_i \sim [r(x)0, w(x)1, r(y)0 \Rightarrow T_j \sim [\leftarrow r(x)1]]$$

- If $T_j$ eventually commits, then the sequential witness history $\hat{S} = \langle T_i, T_j \rangle$ is illegal.
Virtual world consistency

If $T_i$ releases early in $H$, then $T_i$ cannot abort.

$$T_i \left[ \begin{array}{c} r(x)0, w(x)1, r(y)0 \\ \end{array} \right]$$

$$T_j \left[ \begin{array}{c} \sim r(x)1 \\ \end{array} \right]$$

- If $T_j$ eventually commits, then the sequential witness history $\hat{S} = \langle T_i, T_j \rangle$ is illegal.
Virtual world consistency

If $T_i$ releases early in $H$, then $T_i$ cannot abort.

$$T_i \left[ r(x)0, w(x)1, r(y)0 \right] \Rightarrow$$
$$T_j \left[ r(x)1 \right] \Rightarrow$$

- If $T_j$ eventually commits, then the sequential witness history $\hat{S} = \langle T_i, T_j \rangle$ is illegal.
- If $T_j$ eventually aborts, its causal past $C(T_j) = \langle T_i, T_j \rangle$ contains two aborted transactions, so it is illegal.
- Serializability
  - supports early release
  - very basic
- Serializability
  - supports early release
  - very basic
- Opacity—no early release
- TMS1 & TMS2—no early release
- Serializability
  - supports early release
  - very basic
- Opacity—no early release
- TMS1 & TMS2—no early release
- Virtual world consistency
  - supports early release
  - transactions cannot abort
- Serializability
  - supports early release
  - very basic

- Opacity—no early release

- TMS1 & TMS2—no early release

- Virtual world consistency
  - supports early release
  - transactions cannot abort

- Elastic opacity
  - supports early release
  - unintuitive cutting rules
Database properties

- Recoverability
- Avoiding Cascading Aborts
- Strictness
- Rigorousness
Database properties

- Recoverability
  
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- Avoiding Cascading Aborts

- Strictness

- Rigorousness
Database properties

- Recoverability ✓
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- Avoiding Cascading Aborts
- Strictness
- Rigorousness
Database properties

- **Recoverability ✓**
  
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- **Avoiding Cascading Aborts**
  
  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- **Strictness**

- **Rigorousness**
Database properties

- Recoverability ✓
  
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- Avoiding Cascading Aborts $\approx$

  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- Strictness

- Rigorousness
Database properties

- **Recoverability ✓**
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- **Avoiding Cascading Aborts ≈**
  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- **Strictness**
  History $H$ is strict iff for any $T_i, T_j \in H$ and given any operation $op_i = r(x)v$ or $w(x)v'$ in $H|T_i$, and any operation $op_j = w(x)v$ in $H|T_j$, if $op_j$ follows $op_i$, then $T_j$ commits or aborts before $op_i$.

- **Rigorousness**
Database properties

- Recoverability ✓
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- Avoiding Cascading Aborts ≈
  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- Strictness ✗
  History $H$ is strict iff for any $T_i, T_j \in H$ and given any operation $op_i = r(x)v$ or $w(x)v'$ in $H|T_i$, and any operation $op_j = w(x)v$ in $H|T_j$, if $op_i$ follows $op_j$, then $T_j$ commits or aborts before $op_i$.

- Rigorousness
Database properties

- **Recoverability ✓**
  
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- **Avoiding Cascading Aborts ≈**
  
  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- **Strictness ✗**
  
  History $H$ is strict iff for any $T_i, T_j \in H$ and given any operation $o_{i} = r(x)v$ or $w(x)v'$ in $H|T_i$, and any operation $o_{j} = w(x)v$ in $H|T_j$, if $o_{i}$ follows $o_{j}$, then $T_j$ commits or aborts before $o_{i}$.

- **Rigorousness**
  
  History $H$ is rigorous if it is strict and for any $T_i, T_j \in H$ such that $T_j$ writes to variable $x$, i.e., $o_{j} = w(x)v \in H|T_j$ after $T_i$ reads $x$, then $T_i$ commits or aborts before $o_{j}$.
Database properties

- **Recoverability ✓**
  
  History $H$ is recoverable iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits in $H$ before $T_j$.

- **Avoiding Cascading Aborts ≈**
  
  History $H$ Avoids Cascading Aborts iff for any $T_i, T_j \in H$ s.t. $T_j$ reads from $T_i$, $T_i$ commits before the read.

- **Strictness ×**
  
  History $H$ is strict iff for any $T_i, T_j \in H$ and given any operation $op_i = r(x)v$ or $w(x)v'$ in $H|T_i$, and any operation $op_j = w(x)v$ in $H|T_j$, if $op_i$ follows $op_j$, then $T_j$ commits or aborts before $op_i$.

- **Rigorousness ×**
  
  History $H$ is rigorous if it is strict and for any $T_i, T_j \in H$ such that $T_j$ writes to variable $x$, i.e., $op_j = w(x)v \in H|T_j$ after $T_i$ reads $x$, then $T_i$ commits or aborts before $op_j$. 
Serializability + spectrum
Serializability spectrum
Serializability + spectrum

serializability + recoverability

serializability + ACA

serializability
Serializability + spectrum

serializability + recoverability

serializability + ACA

serializability

?
Last-use

**Definition (Commit-pending–equivalence)**

A live transaction $T_i$ in $H$ is commit-pending–equivalent with respect to $x$ iff it is finished executing all of its operations on $x$. 

```java
atomic{
    int v = read(x);
    if (v < 0)
        write(x,-v);
    // commit-pending--equivalent wrt x
    int u = read(y);
    write(y, u + 1);
    // commit-pending--equivalent wrt y
}
```
Definition (Commit-pending–equivalence)

A live transaction $T_i$ in $H$ is commit-pending–equivalent with respect to $x$ iff it is finished executing all of its operations on $x$.

atomic{
    int v = read(x);
    if (v < 0)
        write(x,-v); // commit-pending--equivalent wrt x
    int u = read(y);
    write(y, u + 1); // commit-pending--equivalent wrt y
}
Last-use

**Definition (Commit-pending–equivalence)**

A live transaction $T_i$ in $H$ is commit-pending–equivalent with respect to $x$ iff it is finished executing all of its operations on $x$.

```plaintext
atomic{
    int v = read(x);
    if (v < 0)
        write(x,-v); // commit-pending--equivalent wrt x
    int u = read(y);
    write(y, u + 1); // commit-pending--equivalent wrt y
}
```

**Definition (Early release after last use)**

Transaction $T_i$ releases $x$ after last use in $H$ iff $T_i$ releases $x$ early in $H$ and $T_i$ is commit-pending equivalent wrt $x$. 
Last-use consistency

Definition

- Let $T_{er}^H$ be a subset of all transactions in history $H$ that release some variable early.
- Let $T_{lu}^H$ be a subset of all transactions in history $H$ that release some variable early only after last-use.

History $H$ satisfies last-use consistency if $T_{lu}^H = T_{er}^H$. 
Inconsistent views

Last-use consistency precludes overwriting:

\[
T_i \left[ \begin{array}{c}
w(x)0, \ w(x)1
\end{array} \right]
\]

\[
T_j \left[ r(x)0 \xrightarrow{r} T_j' \right] \left[ \begin{array}{c}
r(x)1, \ w(x)2
\end{array} \right]
\]
Inconsistent views

Last-use consistency precludes overwriting:

\[
T_i \\ [ \ w(x)0, \ w(x)1 \ ] \\
T_j \ \ [ \ \rightsquigarrow r(x)0 \ \rightsquigarrow \ T'_j \ [ \ r(x)1, \ w(x)2 \ ] \\
\]

Allowed inconsistent view:

\[
T_i \ [ \ w(x)0, \ w(x)1 \ \rightsquigarrow \ ] \\\nT_j \ [ \ \rightsquigarrow r(x)0 \ \rightsquigarrow \ T'_j \ [ \ r(x)1, \ w(x)2 \ ] \\
\]
Serializability spectrum

- Serializability + recoverability
- Serializability + ACA

Serializability + recoverability + last-use consistency
Conclusions

- Current safety properties not enough for TM with early release
- Spectrum of database consistency properties
- Last-use consistency
Conclusions

- Current safety properties not enough for TM with early release
- Spectrum of database consistency properties
- Last-use consistency
- Future work: last-use opacity