Weak Persistency Semantics from the Ground Up:

Formalising the Persistency Semantics of ARMv8 & Transactional Models

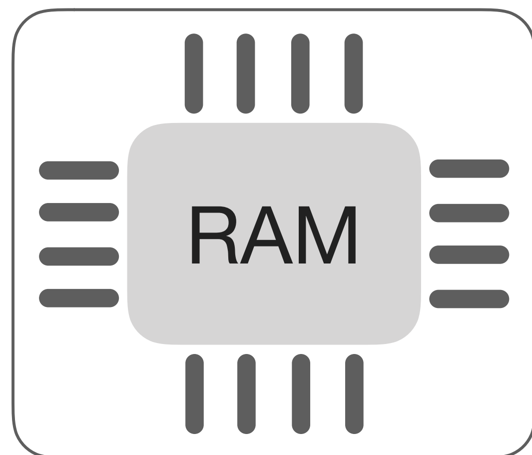
Azalea Raad John Wickerson Viktor Vafeiadis

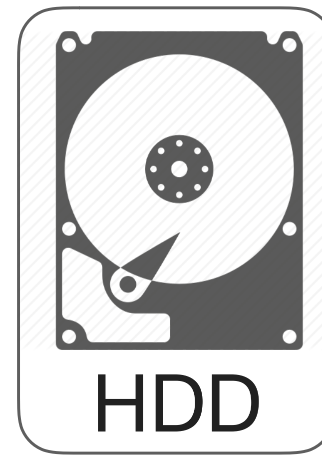
Max Planck Institute for Software Systems (MPI-SWS)
Imperial College London

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

Computer Storage

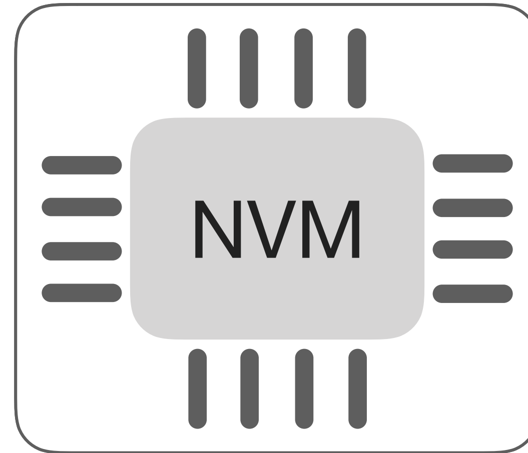
✓ fast
✗ volatile





✗ slow
✓ persistent

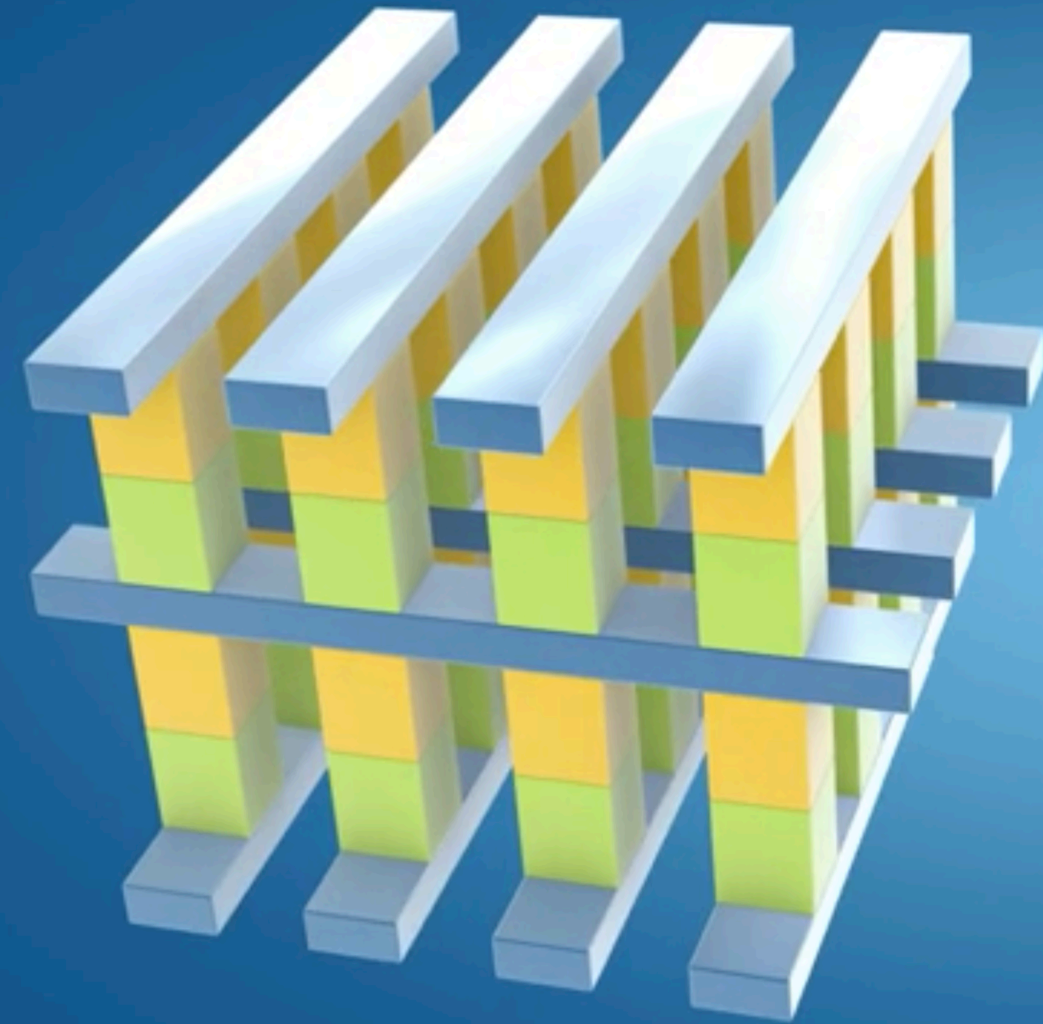
What is Non-Volatile Memory (NVM)?



<u>NVM: Hybrid Storage + Memory</u>

Best of both worlds:

✓ ***persistent*** (like HDD)
✓ ***fast***, ***random access*** (like RAM)



INTEL® OPTANE™ TECHNOLOGY

FAST

DENSE

NON-VOLATILE

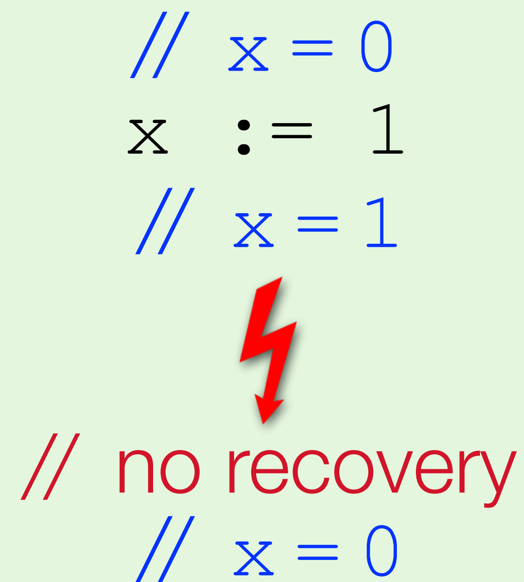
Q: Why ***Formal*** NVM Semantics?

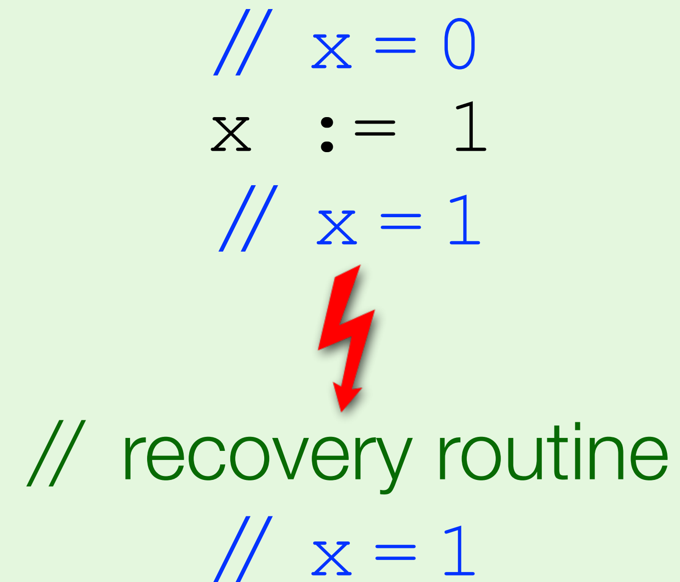
Volatile memory

```
// x = 0
x := 1
// x = 1
```

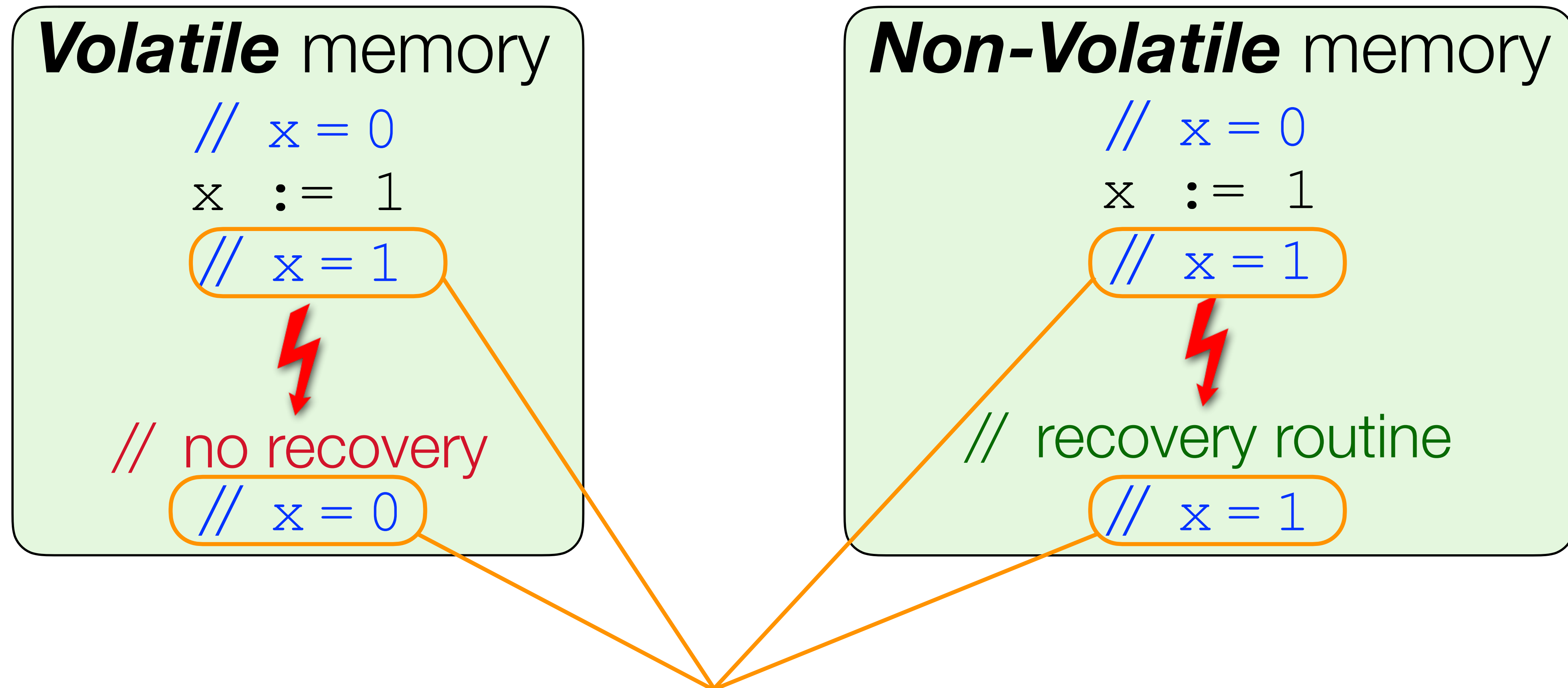
```
// no recovery
// x = 0
```

Q: Why ***Formal*** NVM Semantics?





Q: Why ***Formal*** NVM Semantics?



A: Program ***Verification***

Q: Why ***Formal*** NVM Semantics?

Vol ... ory

What about ***Concurrency***?

```
// x = y = ... = 0
```

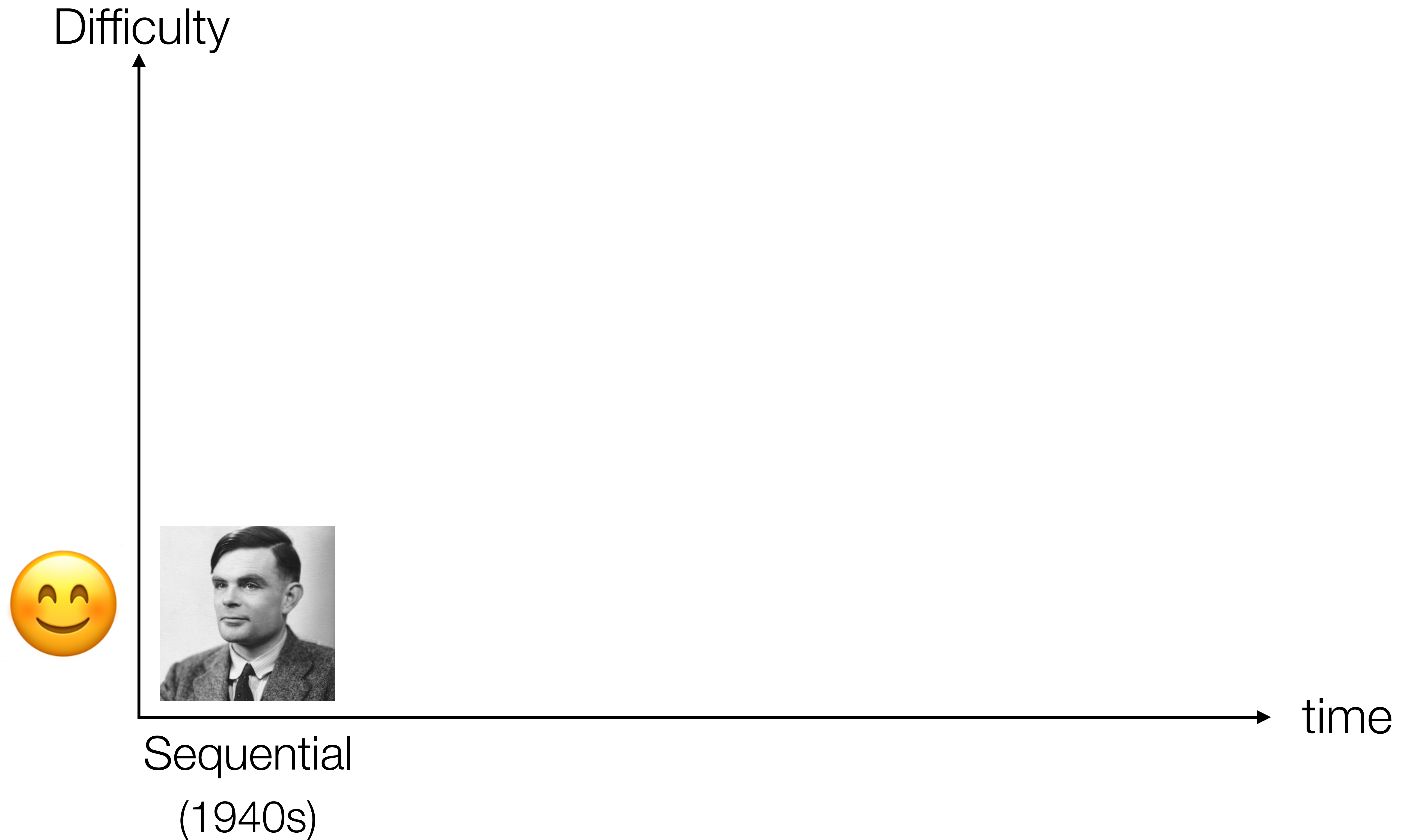
$C_1 \parallel C_2 \parallel \ldots \parallel C_n$

```
// ???
```

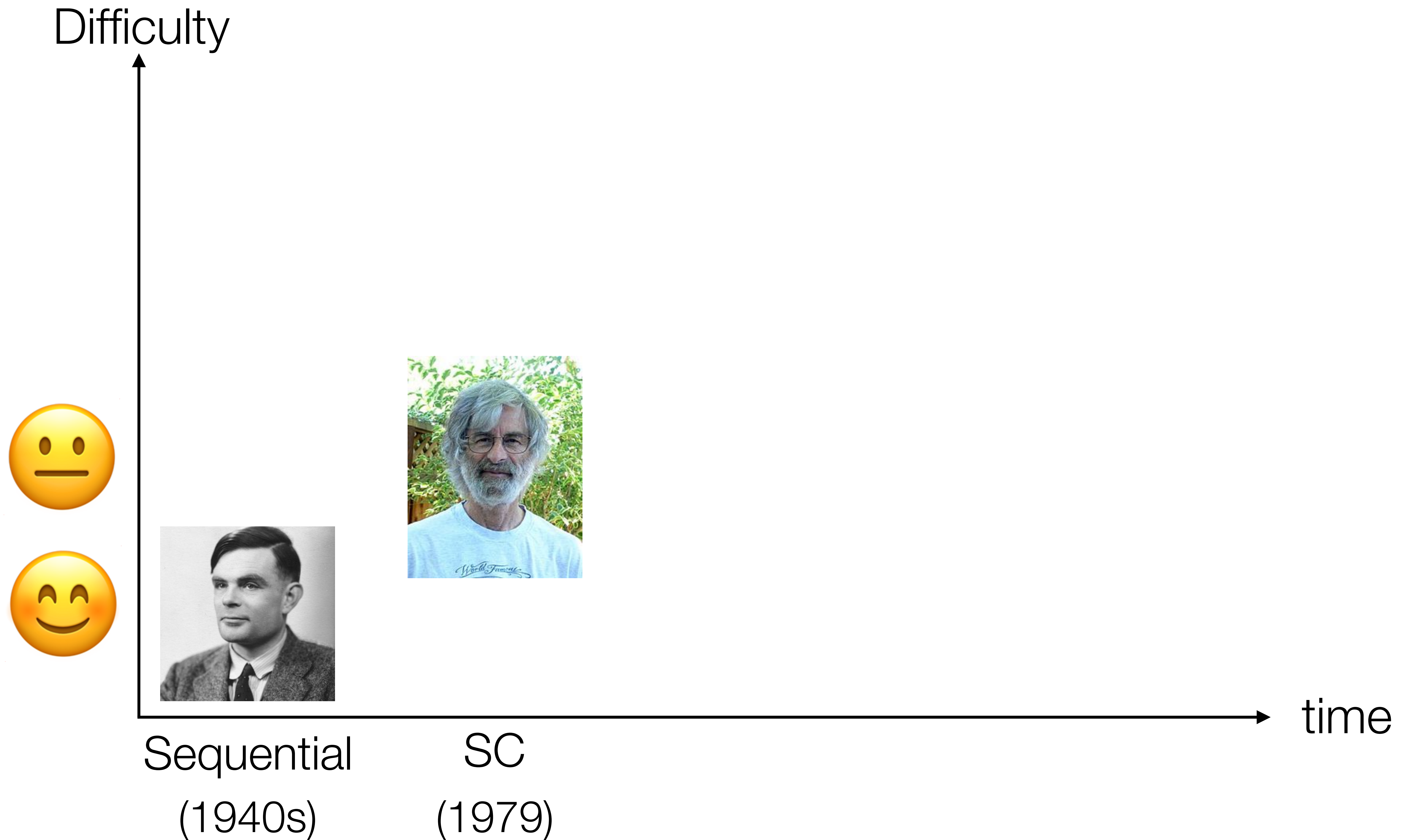
// recovery routine

```
// ???
```

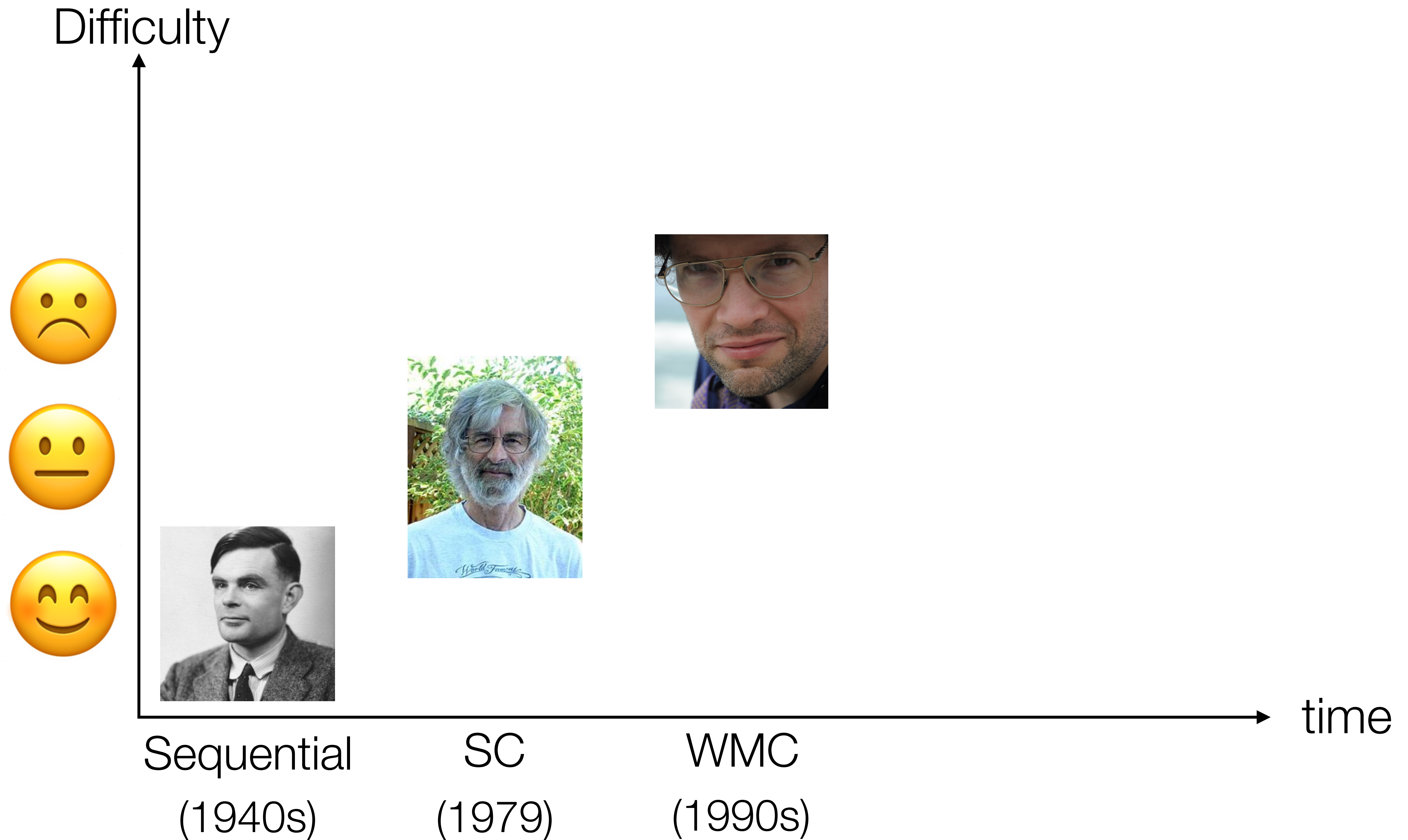
Formal Semantic Models



Formal Semantic Models



Formal Semantic Models



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***

Weak Memory Consistency (WMC)

No total execution order (*to*) ⇒

weak be

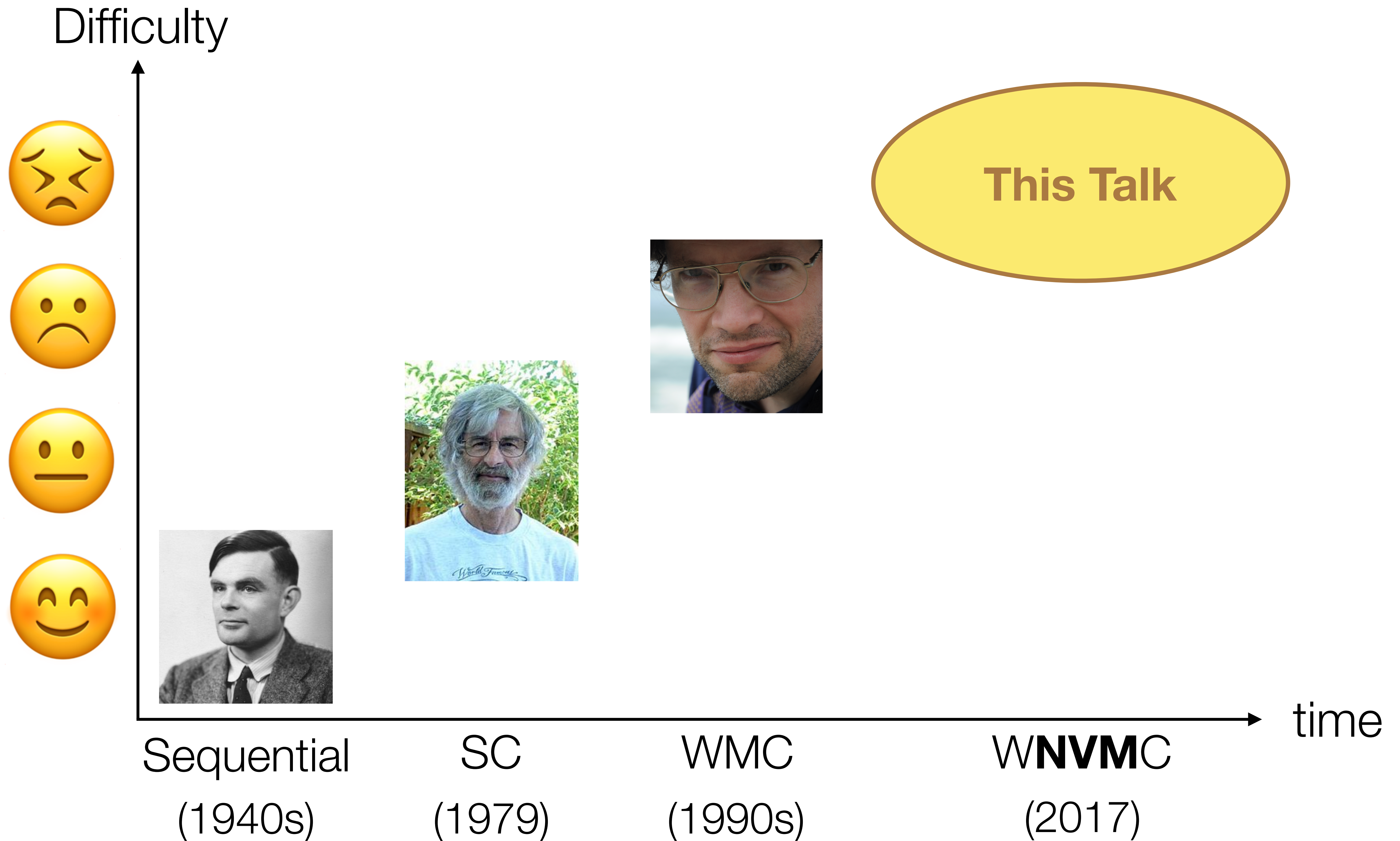
- instru
- write

Consistency Model

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

e.g. TSO, ARMv8, POWER, C11, Java

Formal Semantic Models



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?

// x= ... y=1

!! Ex

!! W

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

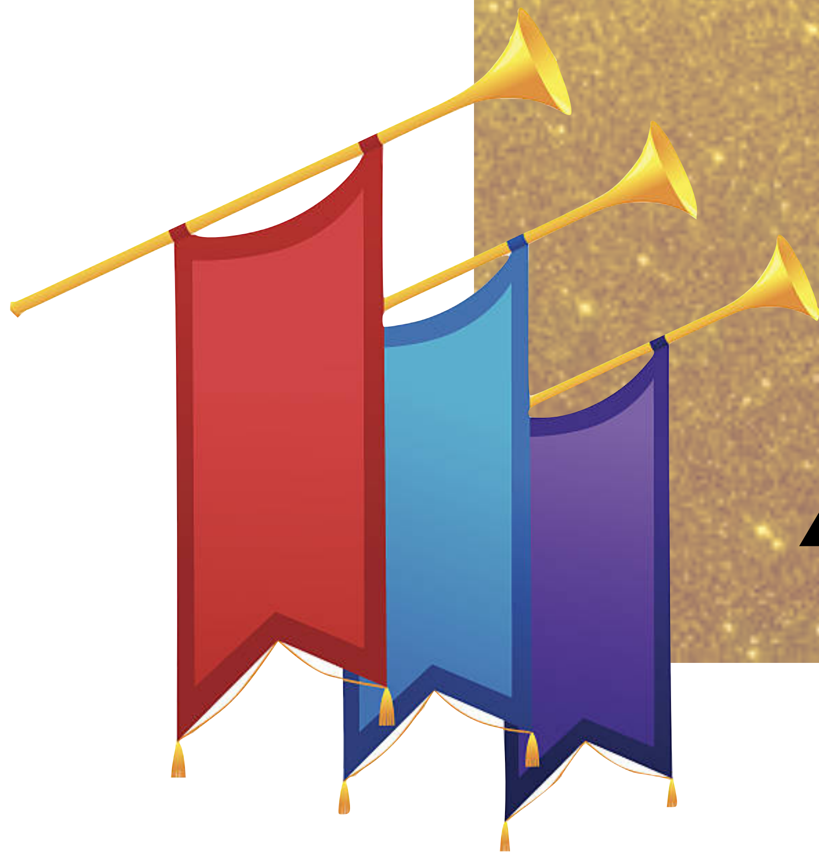
Persistency Model

the ***order*** in which writes
are ***persisted*** to NVM

NVM Semantics

Consistency + Persistency Model

🎊 This Talk 🎊



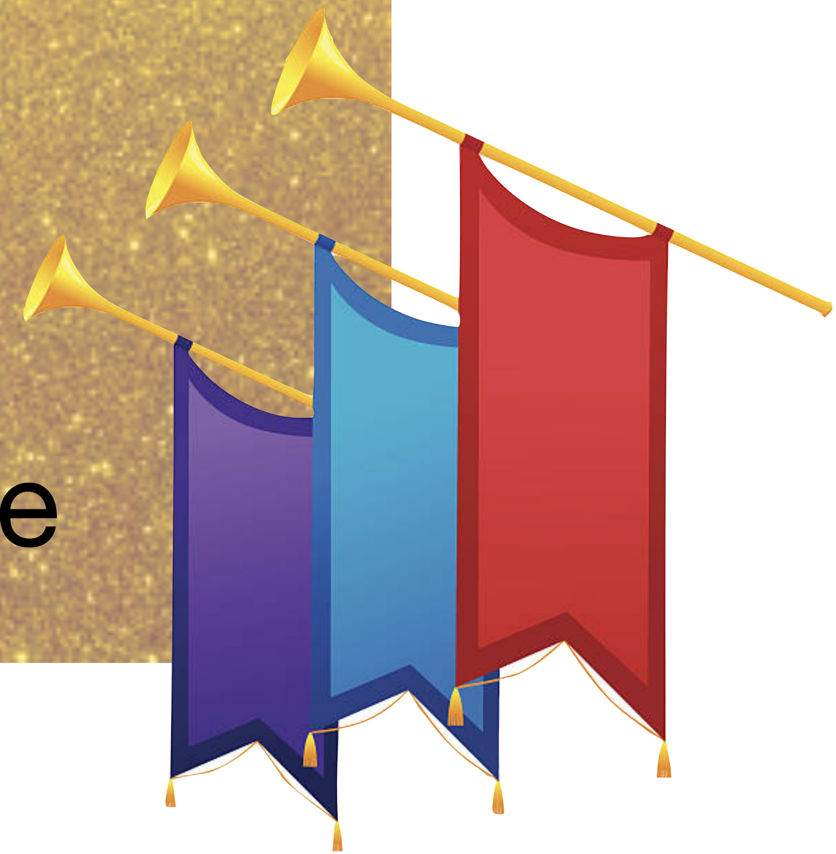
PARMv8

(Persistent ARMv8):

NVM Semantics

of the

ARMv8 Architecture



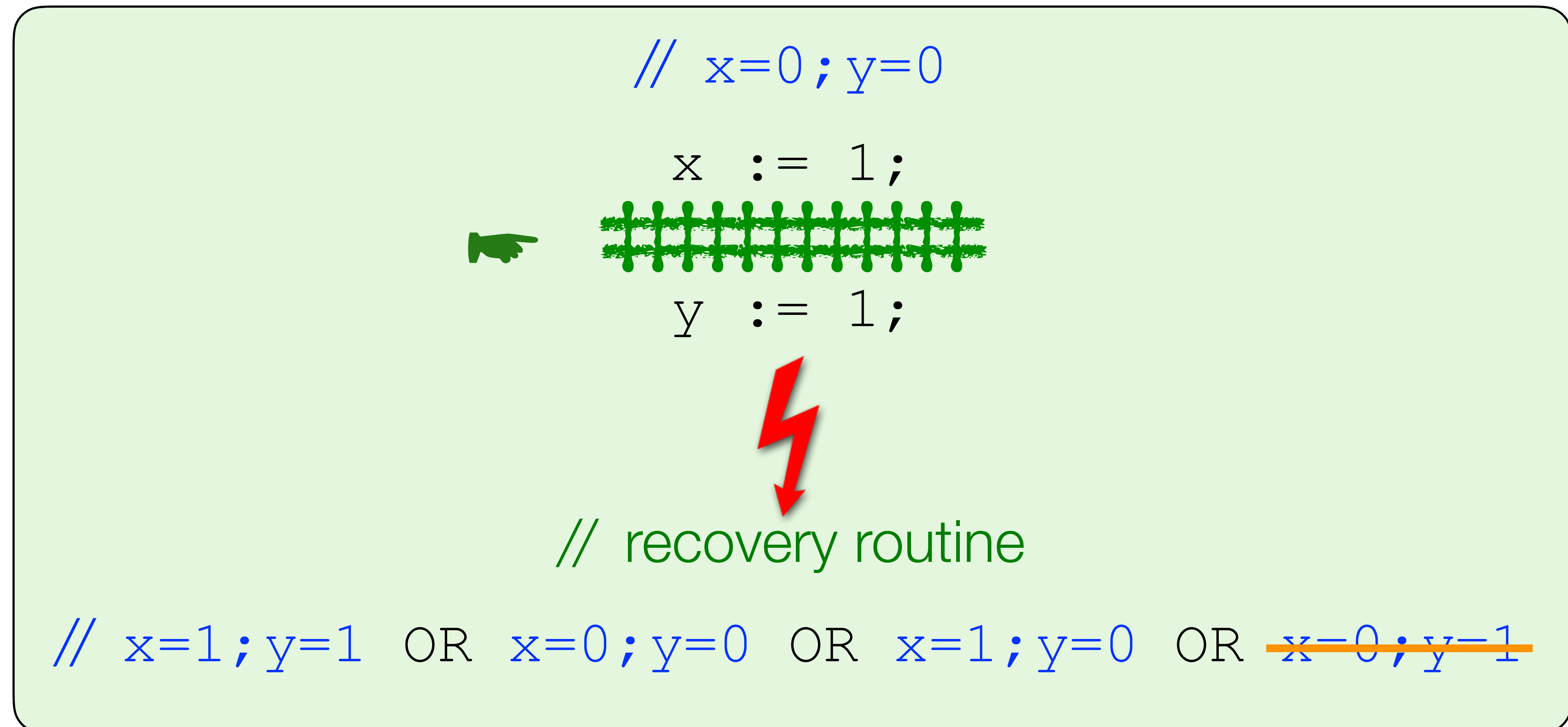
Challenge #1: ***Relaxed*** Persists

```
// 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*



!! ***out of order* persists**

☛ ***persist barriers?***

Persist Barriers: ***Desiderata***

ARMv8

does not provide

persist barriers!

ARMv8 memory barriers

(e.g. **`DSB-full`**)

do not enforce

persist ordering!

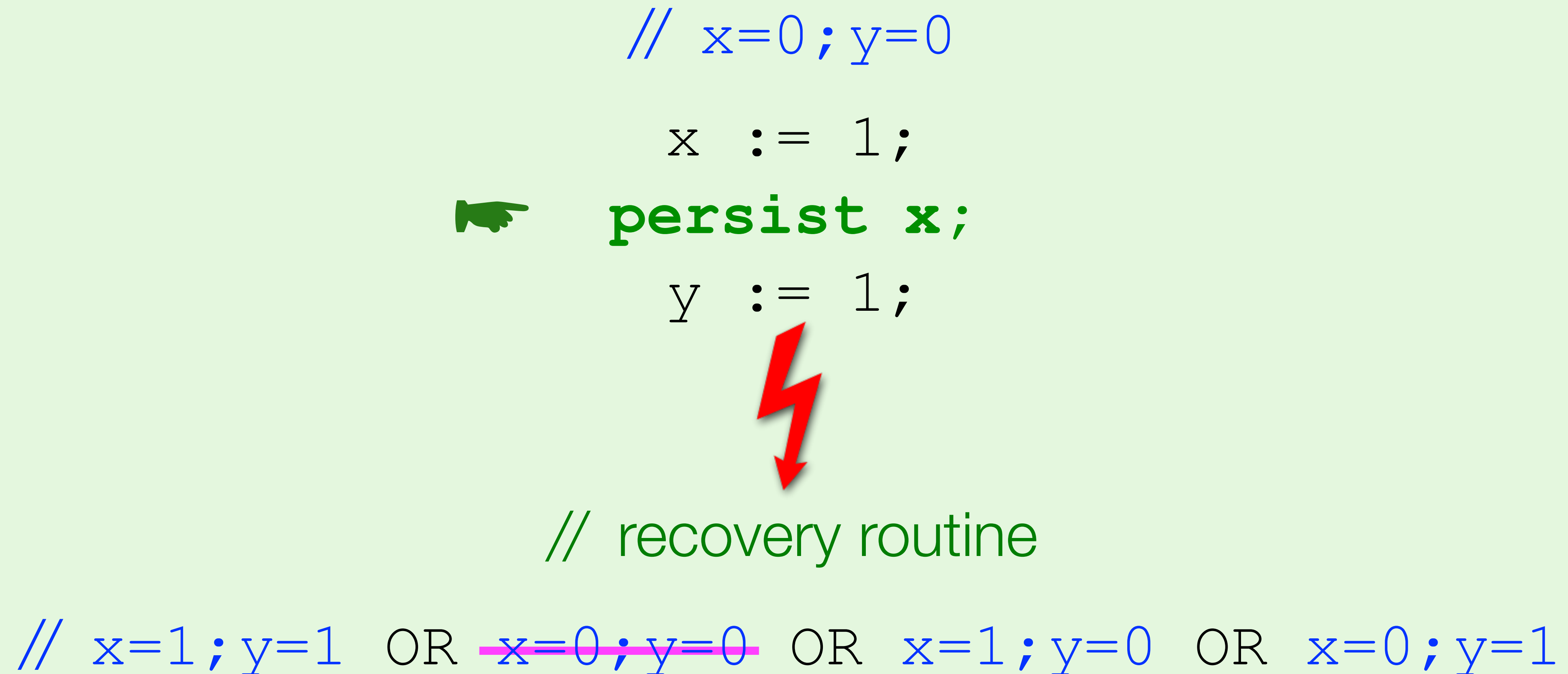
Challenge #2: ***Asynchronous*** Persists

```
// 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***



!! Execution continues ***ahead of persistence***

☛ ***explicit persists?***

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***

☛ ***explicit persists?***

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***

☛ ***explicit persists?***

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

Explicit Persists: ***Reality on ARMv8***

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

x := 1;
☛ DC-CVAP x;

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

ARMv8 explicit persists
are themselves
asynchronous!

!! Execution continues ***ahead of persistence***

☛ ***explicit persists?***

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

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
```

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
```

(The option "x=0;y=0" is struck through.)

✤ ***Waits*** until earlier writes on **x** are persisted

✓ ***synchronous*** persists

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
- ***Disallows reordering***

✓ ***synchronous*** persists

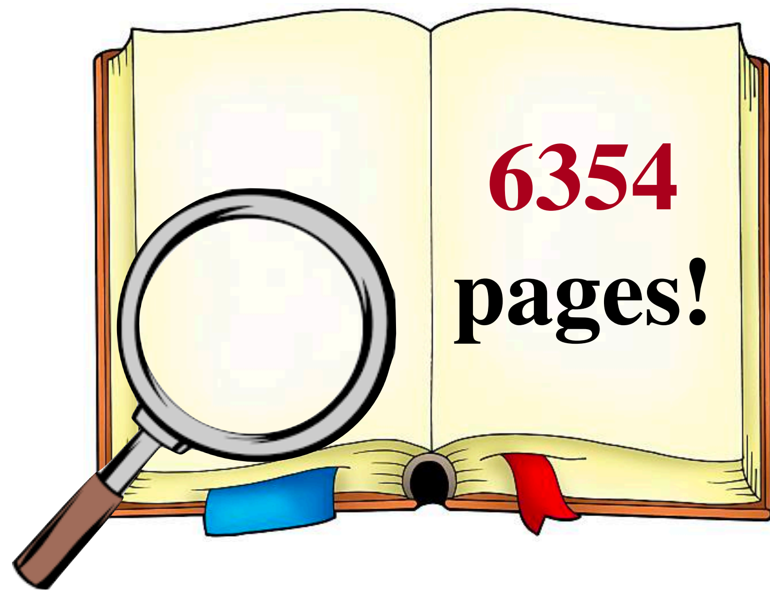
✓ ***no out of order*** persists

PARMv8

ARM® Architecture Reference Manual

PARMv8

ARM® Architecture Reference Manual

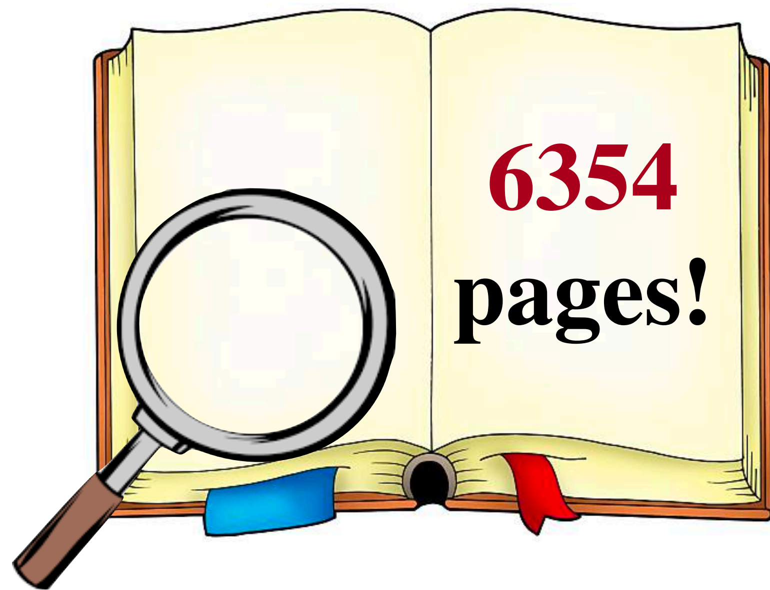


" a `DSB-full` will not complete until all previous `DC-CVAP` have completed "

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

PARMv8

ARM® Architecture Reference Manual



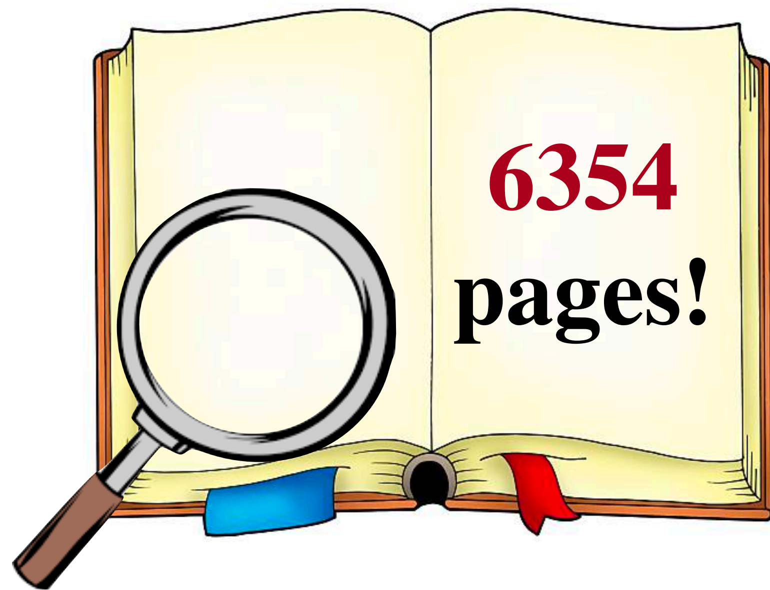
" a **`DSB-full`** will not complete until all previous **`DC-CVAP`** have completed "

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

Ambiguities in text!

PARMv8

ARM® Architecture Reference Manual



" 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!

PARMv8 Axiomatic Specification

- ARMv8 axioms in [Pulte et al. 2018] hold
- $(\mathrm{po}^?; [DMB_{\mathtt{full}} \cup DSB_{\mathtt{full}}]; \mathrm{po}^?) \setminus \mathrm{id} \subseteq \mathrm{ob}$
- $\forall X \in \mathrm{CL}.\ [W_X \cup R_X]; \mathrm{po}; [WB_X] \subseteq \mathrm{ob}$
- $\forall X \in \mathrm{CL}.\ [WB_X]; \mathrm{po}; [WB_X] \subseteq \mathrm{ob}$
- $dom([WB]; \mathrm{ob}; [DSB_{\mathtt{full}}]) \subseteq P$
- $[WB]; \mathrm{ob}; [DSB_{\mathtt{full}}]; \mathrm{ob}; [D] \subseteq \mathrm{nvo}$
- $\forall X \in \mathrm{CL}.\ [W_{X_p}]; \mathrm{ob}; [WB_{X_p}] \subseteq \mathrm{nvo}$
- $\forall x_p \in \mathrm{PLoc}.\ \mathrm{mo}_{x_p} \subseteq \mathrm{nvo}$

PARMv8

Problem

ambiguous text
counter-intuitive semantics
low-level hardware details

Solution

high-level, hardware-agnostic
NVM libraries:

Persistent Transactions

What is a Transaction?

Concurrency control mechanism:

- ***atomic*** work unit:
  - ➡ all-or-nothing writes
- ***consistent*** (e.g. serialisable)

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

// 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)

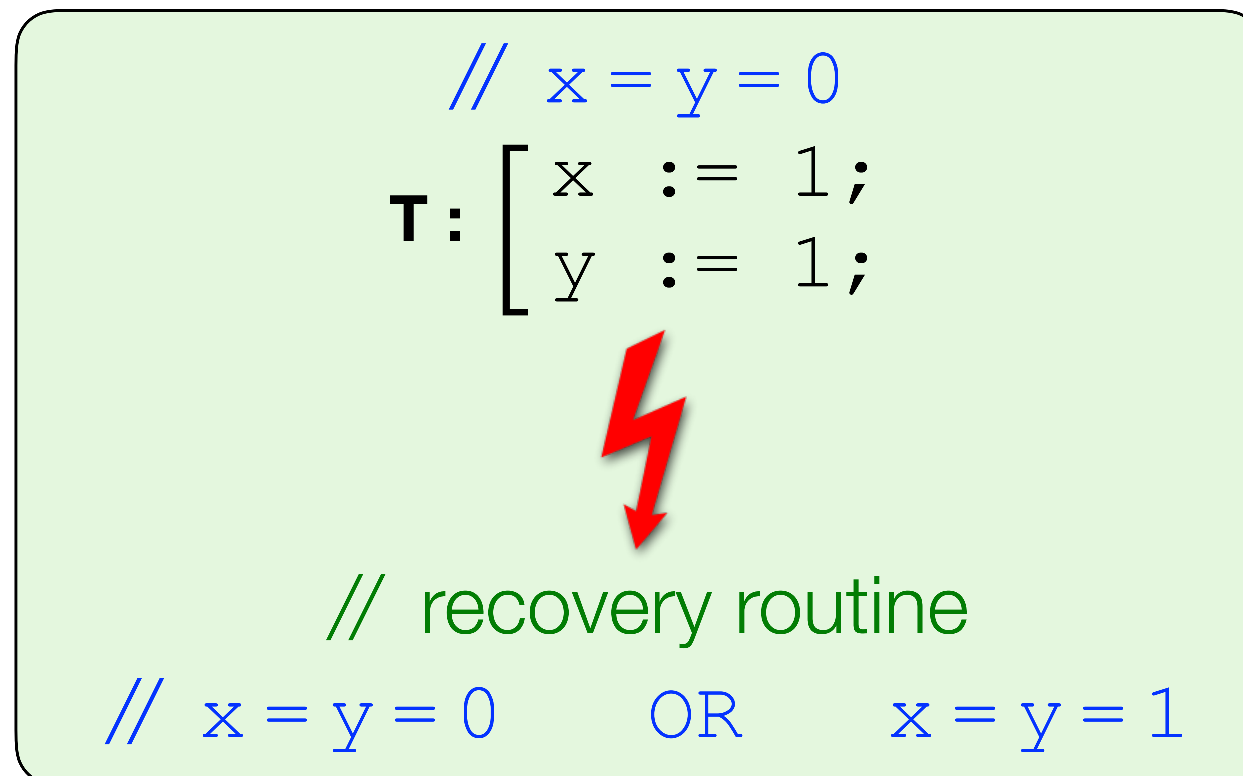
```
// x = y = 0
T: [ x := 1;
     y := 1;

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

What is a ***Persistent*** Transaction?

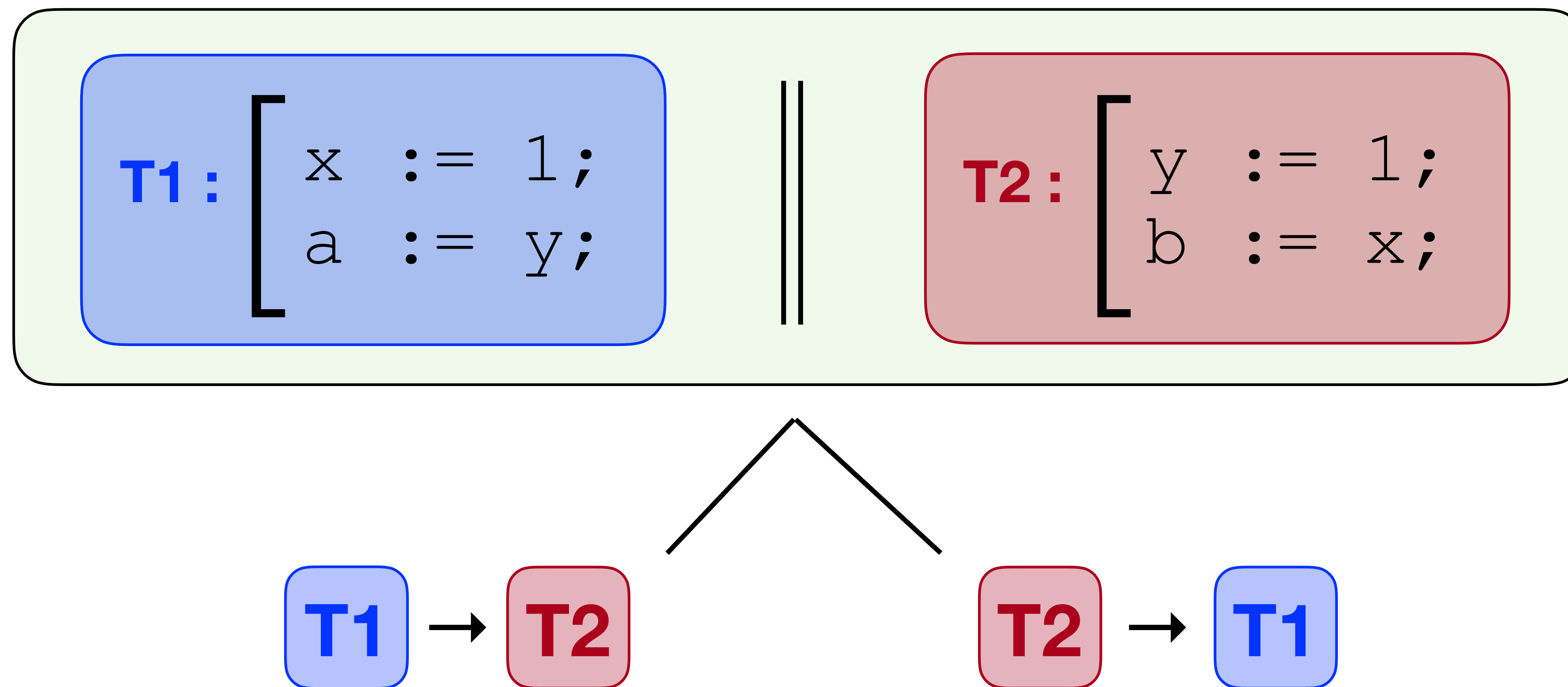
Concurrency & ***persistency*** control mechanism:

- ***atomic*** work unit:
  - ➡ all-or-nothing writes
  - ➡ all-or-nothing ***persists***
- ***consistent*** (e.g. serialisable)
- ***persistent*** (e.g. ***persistently serialisable***)



Serialisability (SER)

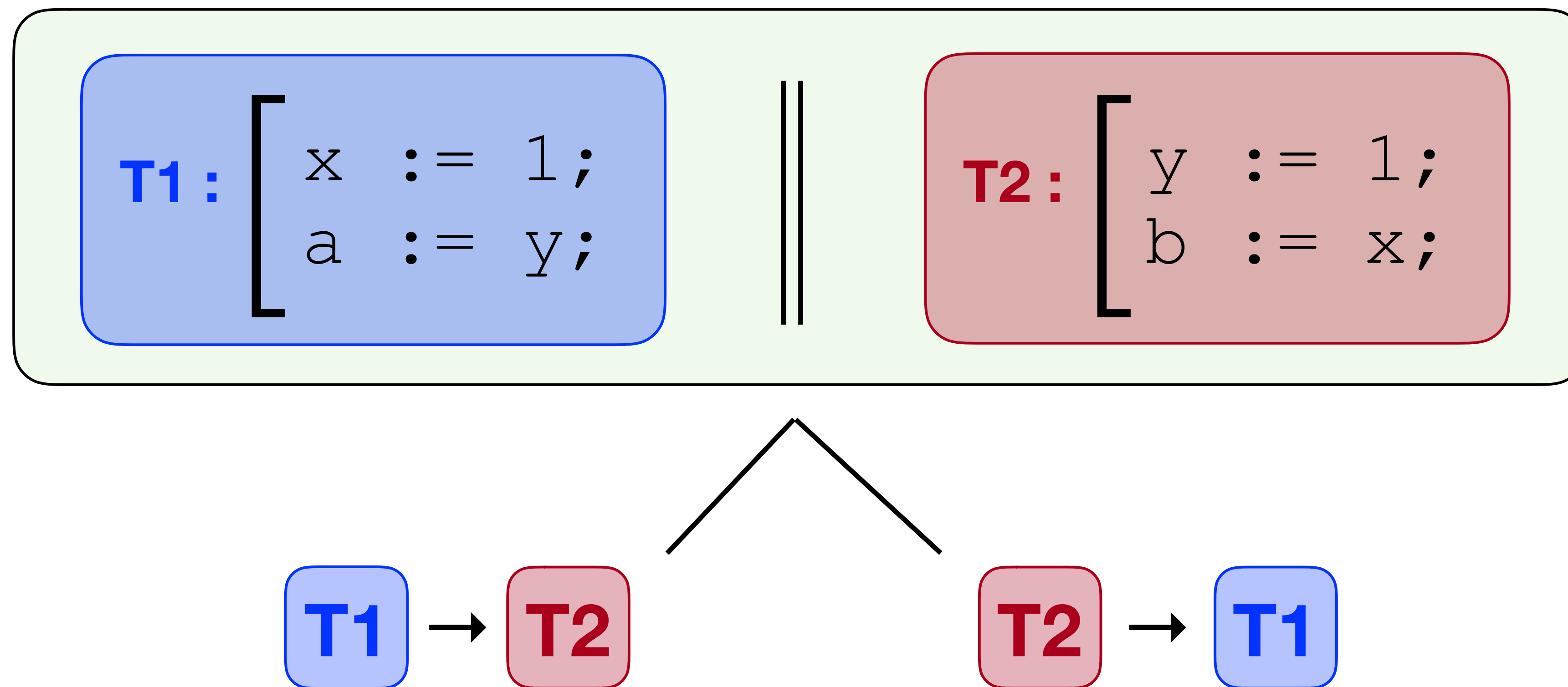
All transactions appear to ***execute*** in a sequential order



Persistent Serialisability (PSER)

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

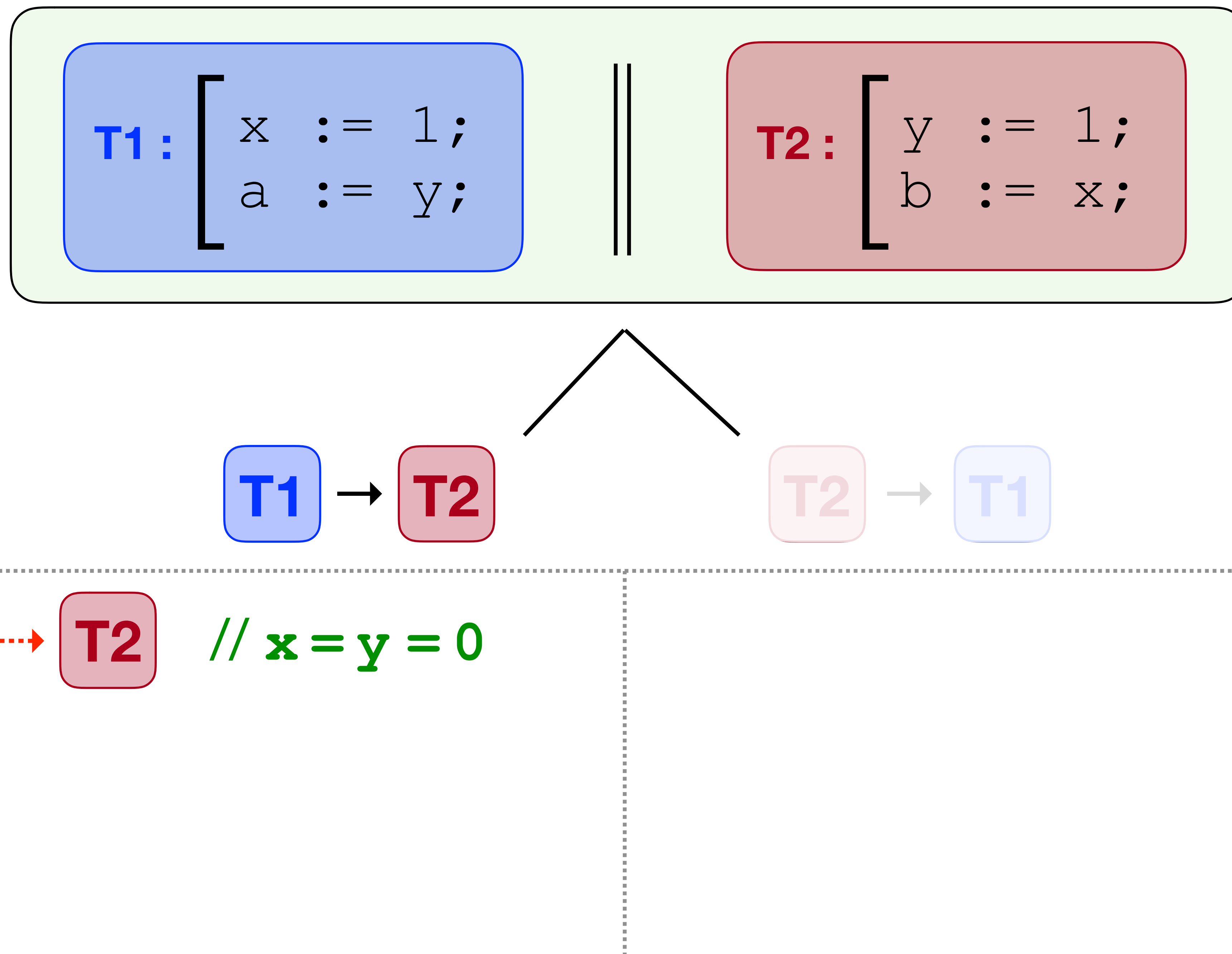
A prefix of transactions appears to ***persist*** in the ***same sequential order***



Persistent Serialisability (PSER)

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

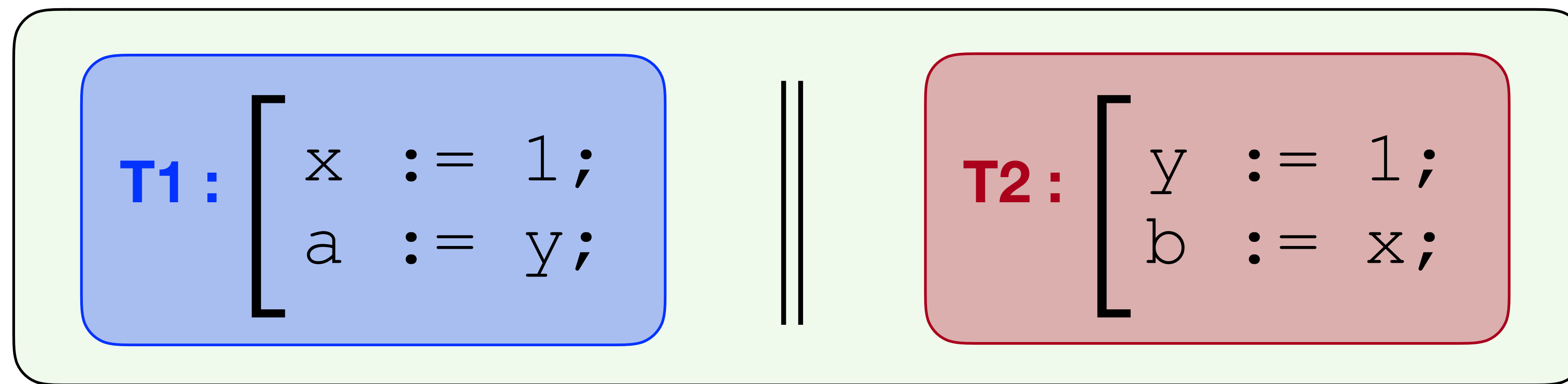
A prefix of transactions appears to ***persist*** in the ***same sequential order***

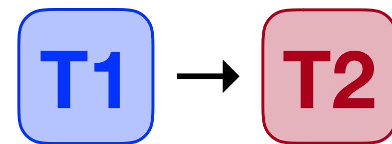


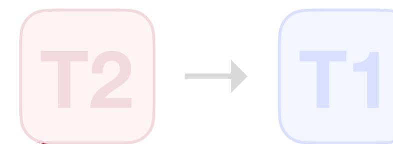
Persistent Serialisability (PSER)

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

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



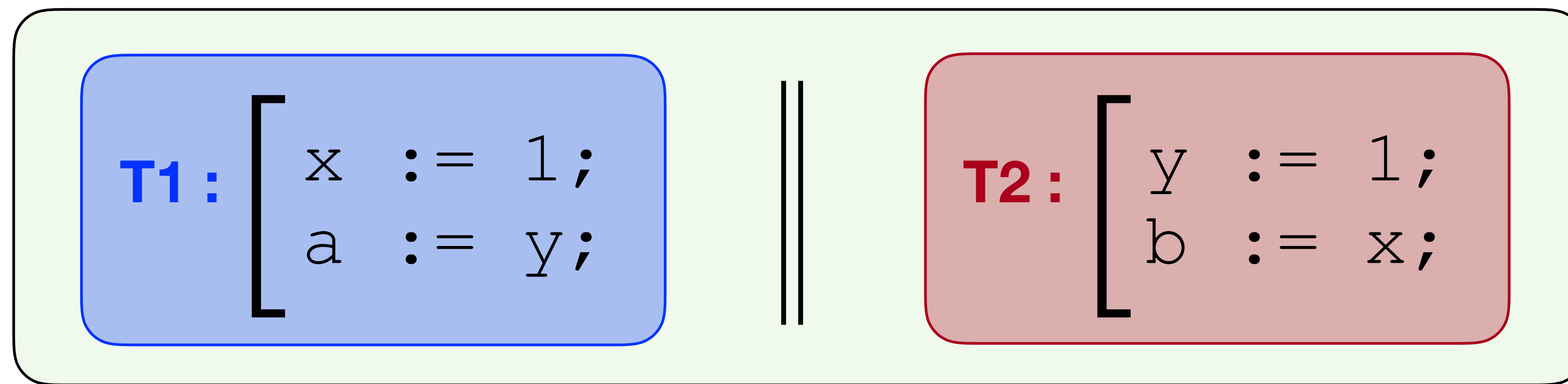


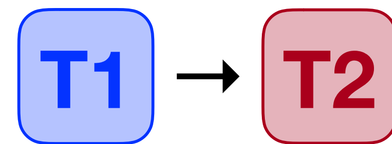


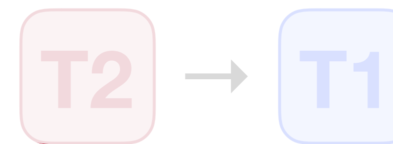
Persistent Serialisability (PSER)

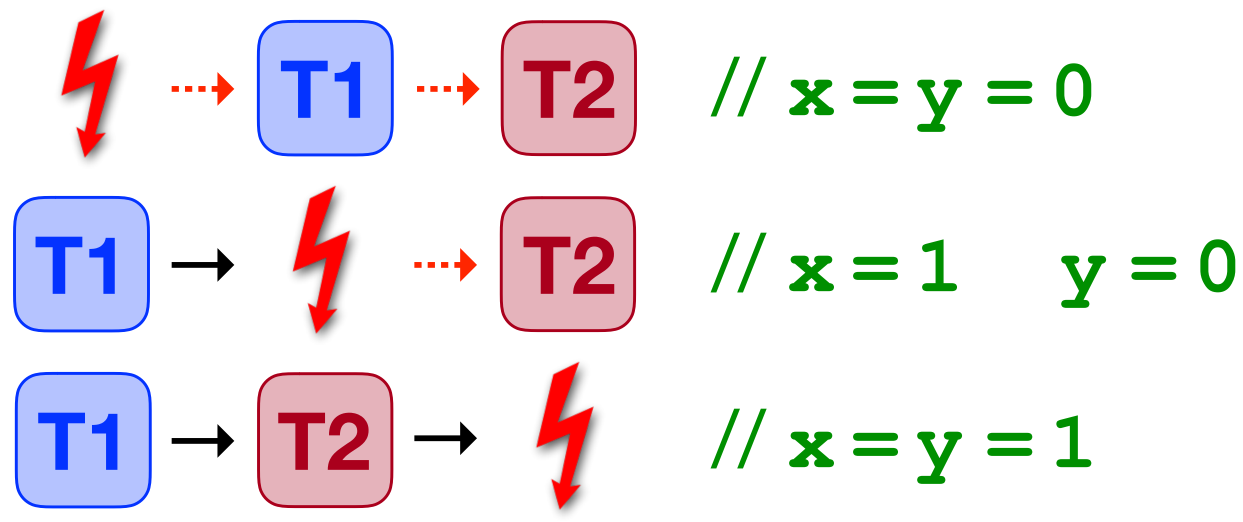
All transactions appear to ***execute*** in a sequential order

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





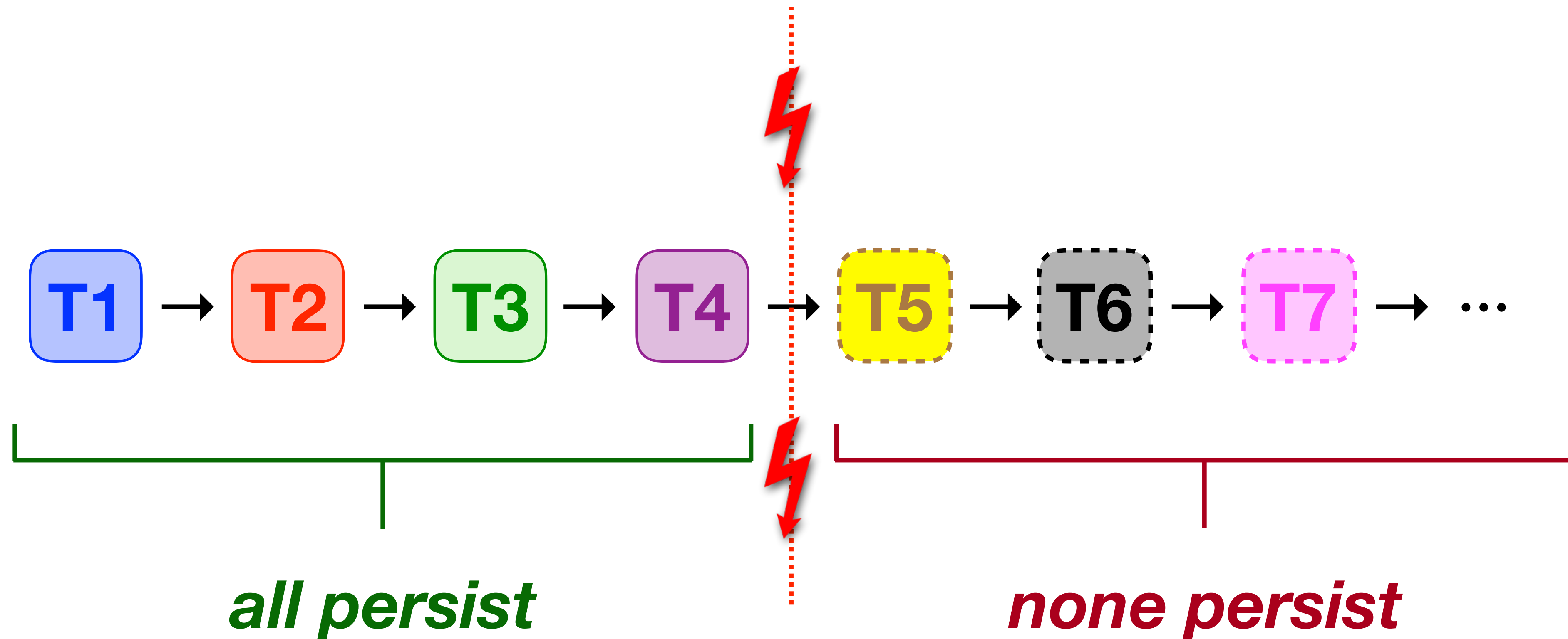




Persistent Serialisability (PSER)

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

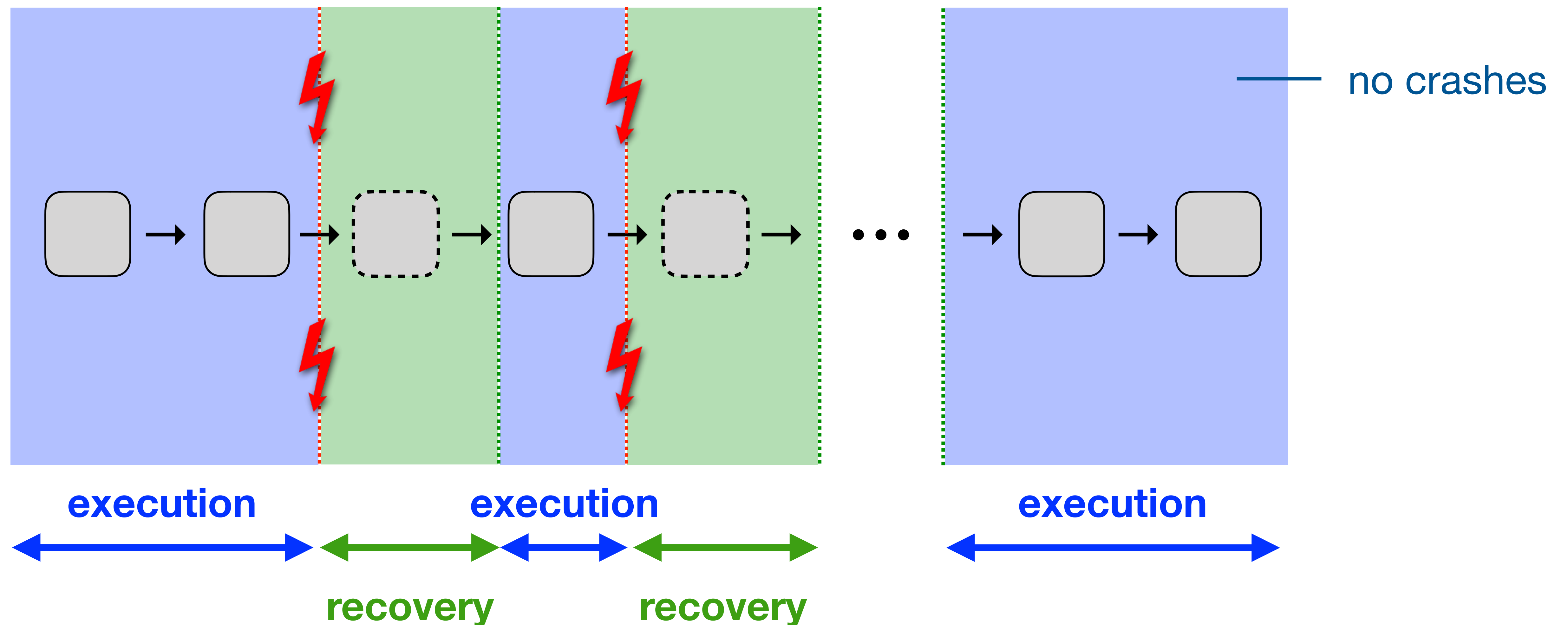
A prefix of transactions appears to ***persist*** in the ***same sequential order***



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***



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

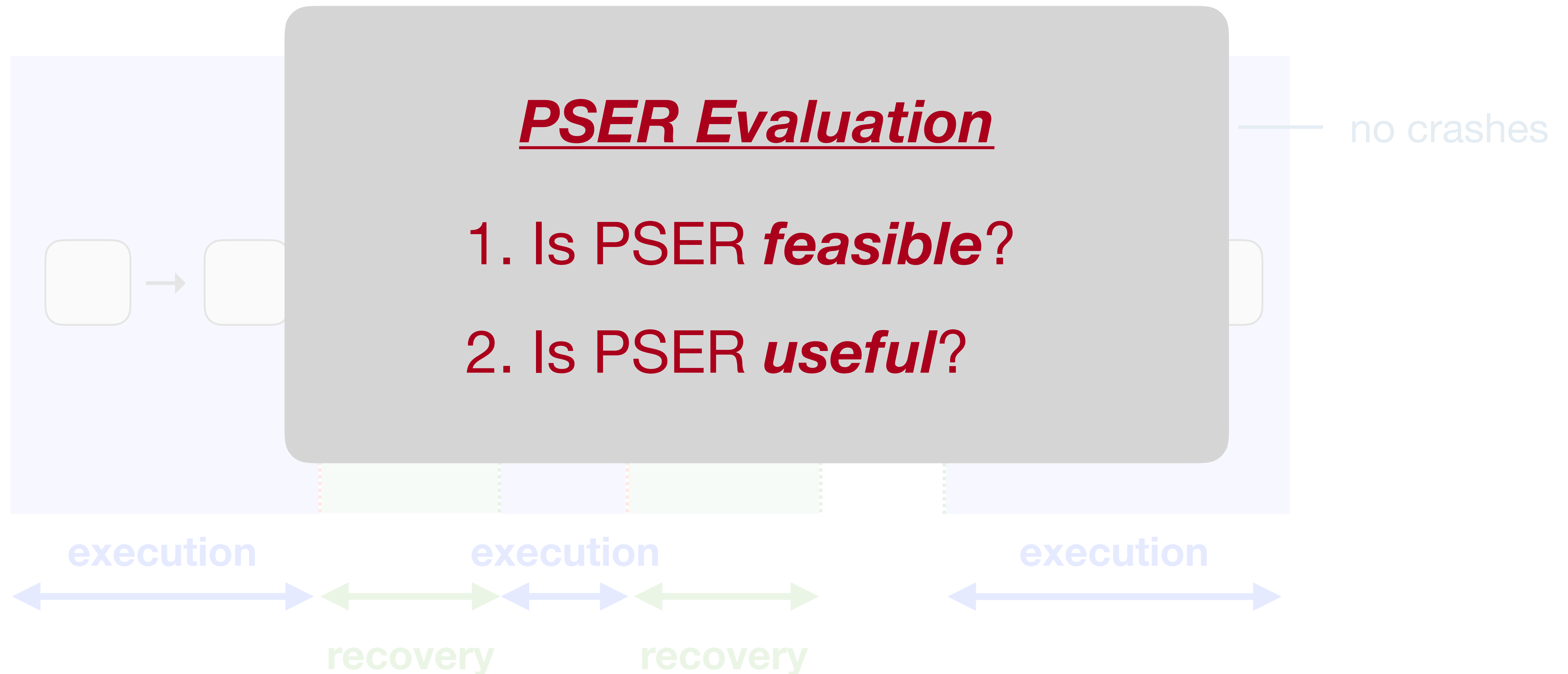
Strong guarantees

Intuitive semantics

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***



Is PSER ***Feasible***?

✓ PSER ***implementation*** in ***ARM***

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

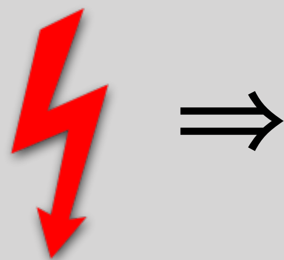
- add code for ***persistence*** — i.e. persist sequences
- add code to ***log metadata*** for ***recovery***
- add ***recovery mechanism***

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

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

deq(q)=
  <deq_body>
```

sequential queue imp.

```
enq(q,v)=
  pser{
    <enq_body> }

deq(q)=
  pser{
    <deq_body> }
```

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

⇒ ***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):

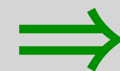
1. T

2. w

n of *L*

Yes!

any correct ***sequential*** implementation



correct, ***concurrent*** & ***persistent***

implementation

```
en

de
    < deq_body >
```

sequential queue imp.

```
    < deq_body > }
```

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!

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