# Specifying & Verifying Non-Volatile Memory

**Azalea Raad**

Max Planck Institute for Software Systems (MPI-SWS)

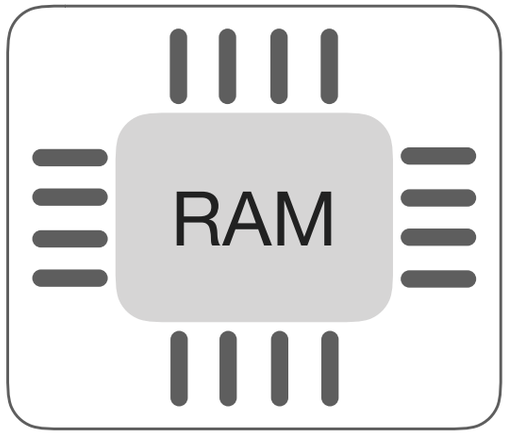
azalea@mpi-sws.org SoundAndComplete.org @azalearaad

# What is ***Non-Volatile Memory* (*NVM*)?**

# Computer Storage

✓ ***fast***

✗ ***volatile***

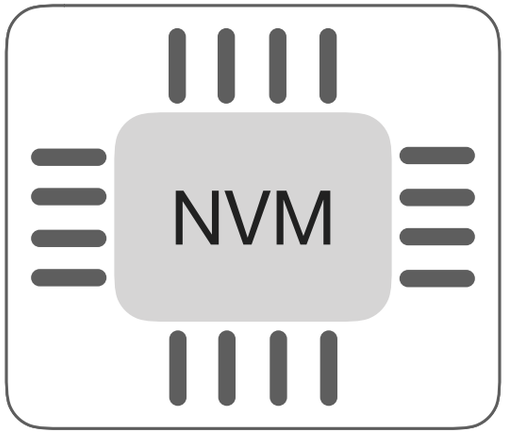




✗ ***slow***

✓ ***persistent***

# What is Non-Volatile Memory (NVM)?



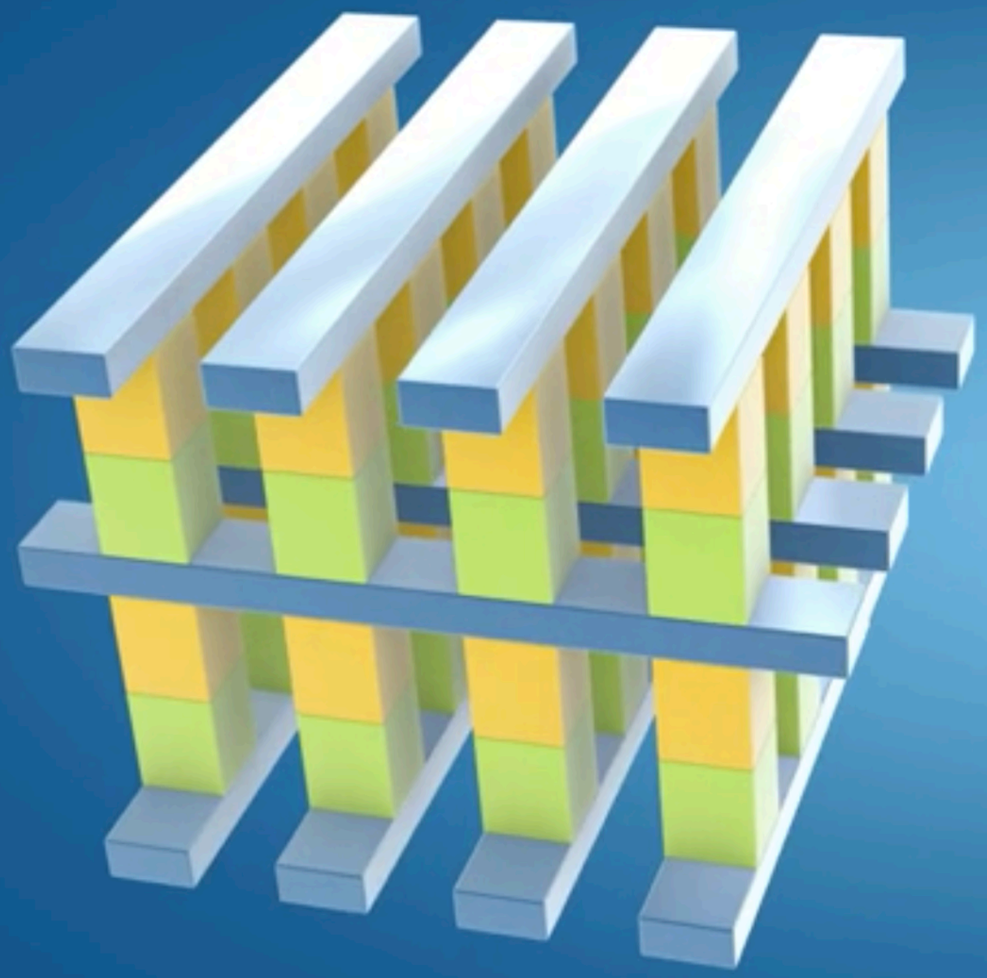

***NVM: Hybrid Storage + Memory***

Best of both worlds:

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

# Why NVM ***Now***?

- ✓ ***Persistent***
- ✓ ***Higher capacity*** (than RAM)
- ✓ ***Green***
  - 32x capacity for 3x power consumption
- ✓ ***Low latency — no intermediaries***
  - no OS/FS intermediaries
  - no paging, no context switching, no interrupts, no kernel code
  - short instruction path
- ✓ Many ***applications***
  - fast in-memory databases
  - file systems
  - …



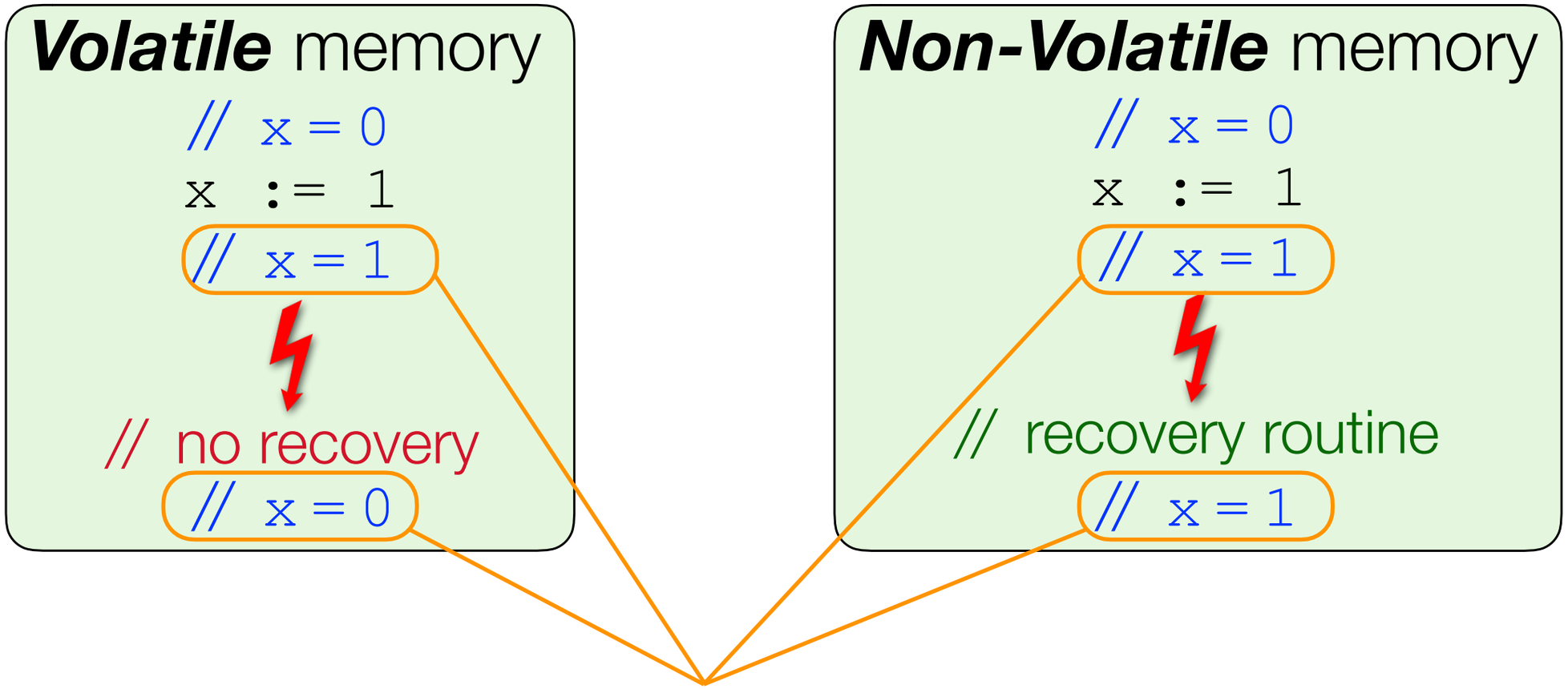
# INTEL® OPTANE™ TECHNOLOGY

FAST

DENSE

NON-VOLATILE

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




# **A**: Program ***Verification***

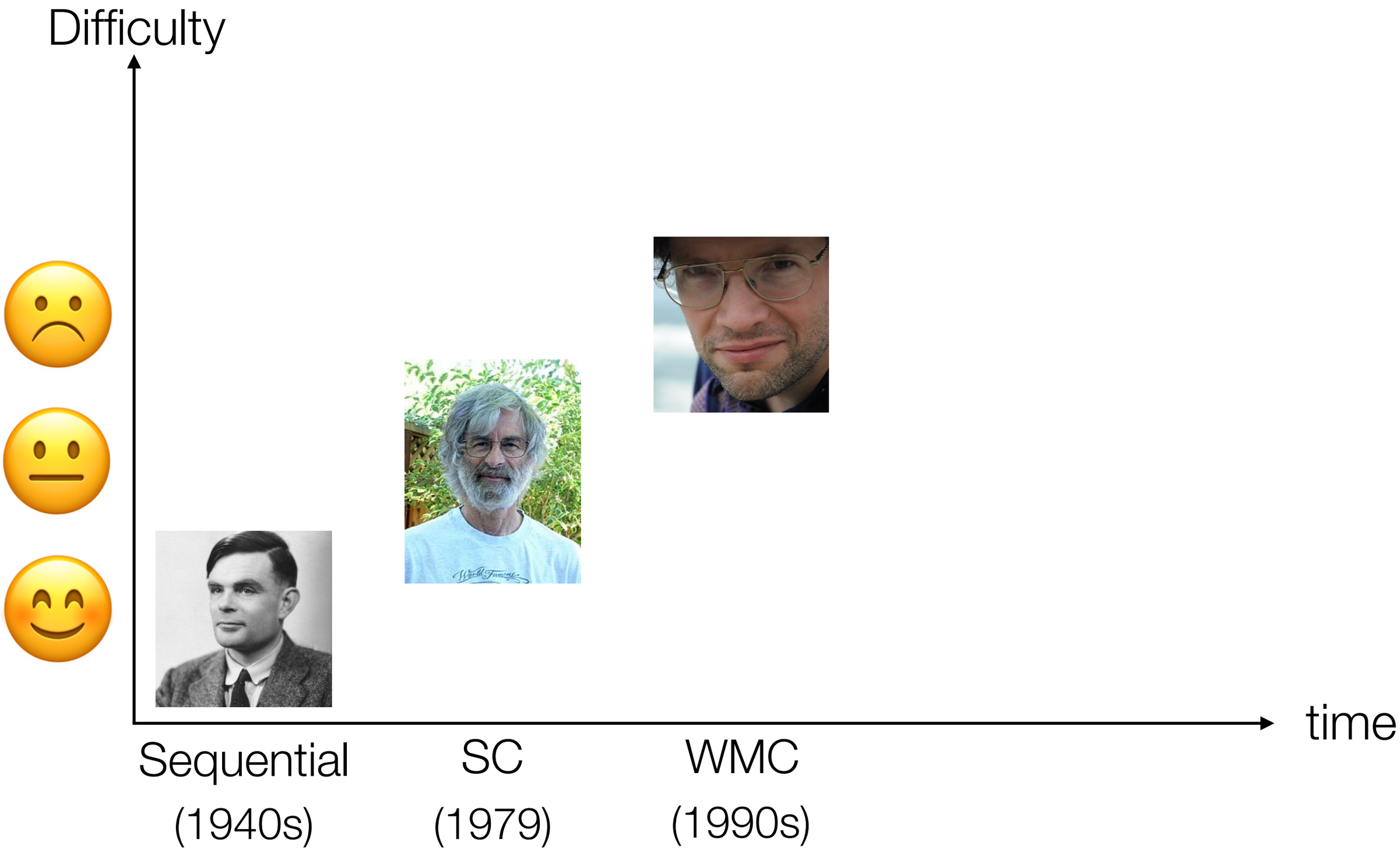
# **Q**: Why ***Formal*** NVM Semantics?

## What about ***Concurrency***?

```
// x = y = ... = 0
C1 || C2 || ... || Cn
// ???
```

```
// recovery routine
// ???
```

# Formal Semantic Models



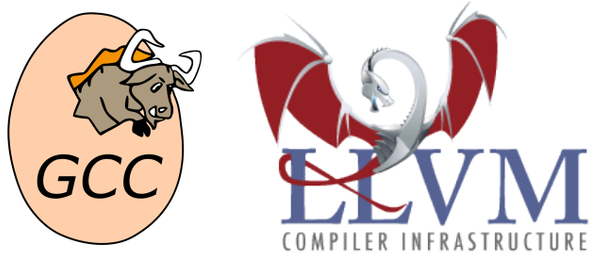
# Weak Memory Consistency (WMC)

***No*** total execution order (*to*) ⇒

***weak*** behaviour absent under SC, caused by:

- **software**

  instruction ***reordering*** by compiler




- **hardware**

  write propagation across ***cache hierarchy***

# Weak Memory Consistency (WMC)

***No*** total execution order (*to*) ⇒

***weak*** be

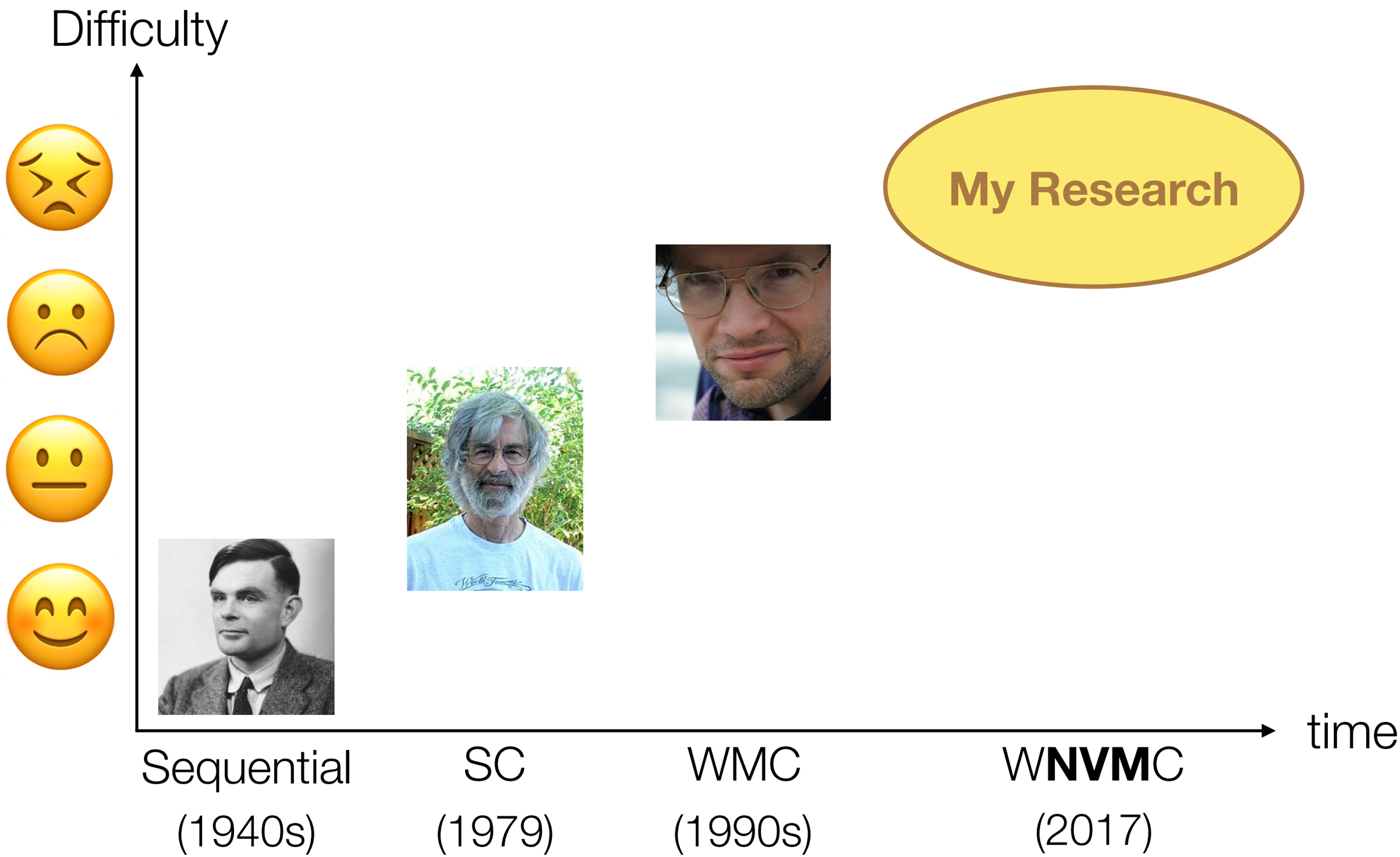
- **soft**
- **hard**

***Consistency Model***

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

e.g. x86-TSO, ARM, POWER, C11, Java

# Formal Semantic Models



# What Can Go Wrong?

```
// 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***

— ***asynchronous*** persists

# What Can Go Wrong?

```
// 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***

— ***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

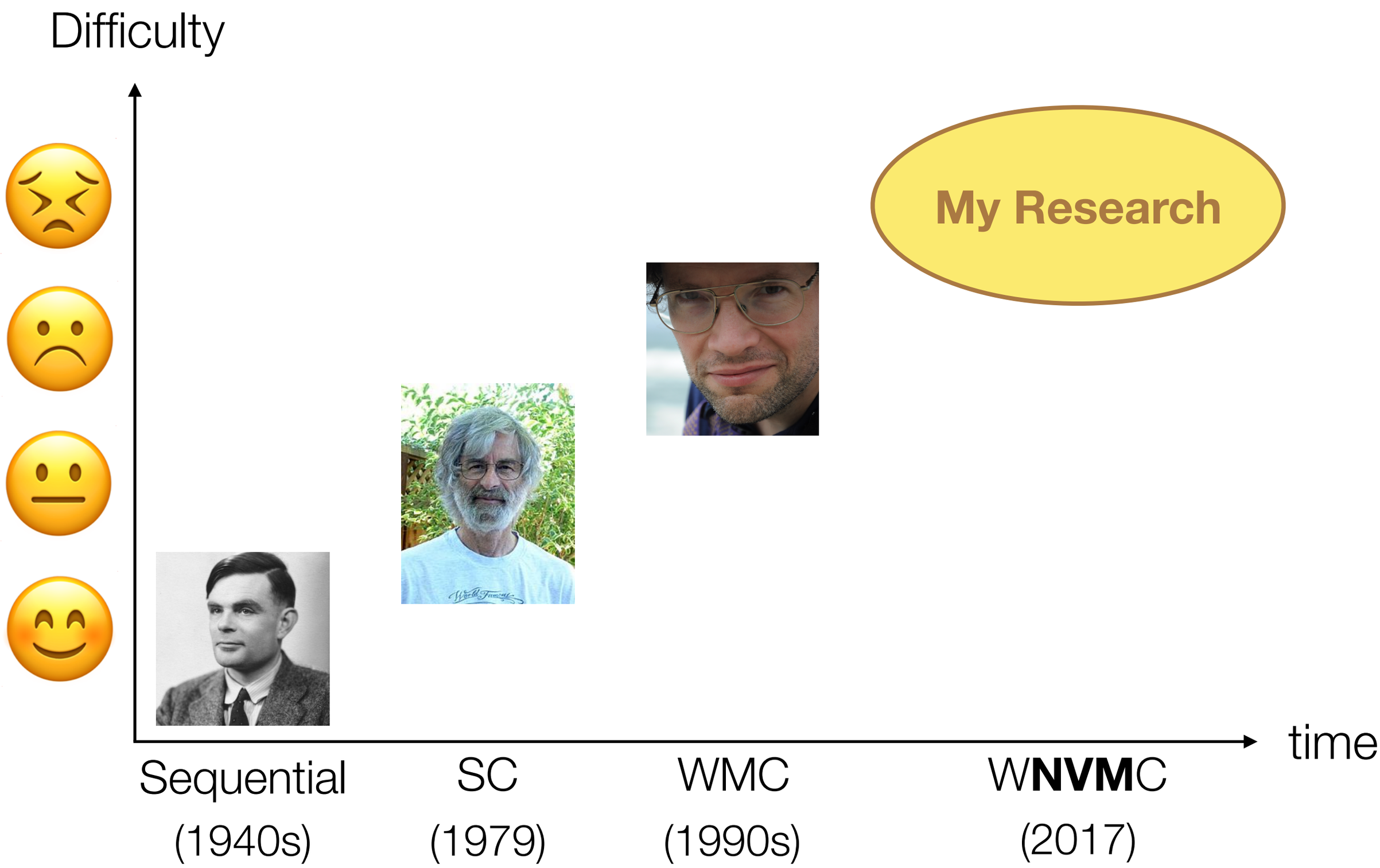
## ***Persistency Model***

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

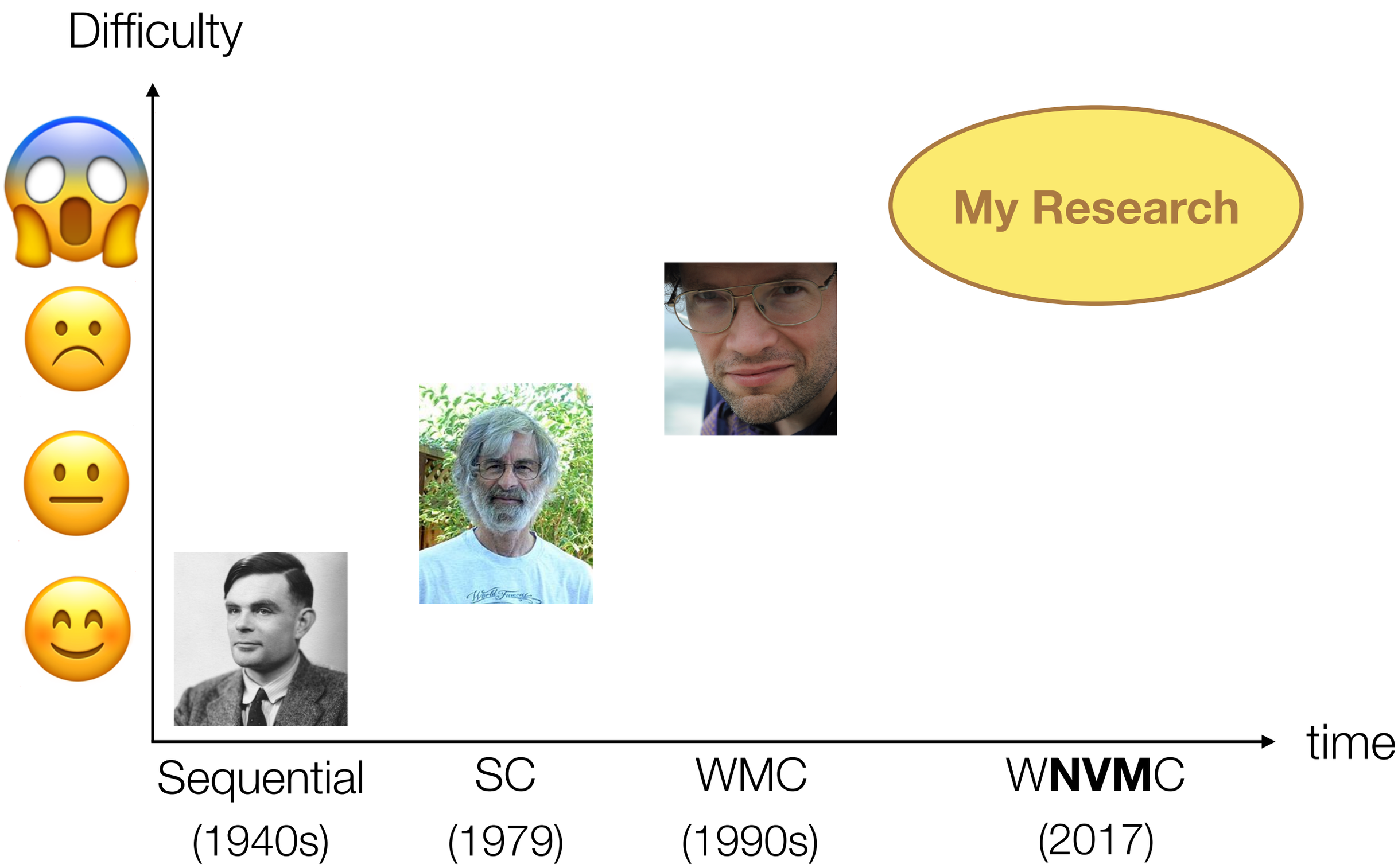
## ***NVM Semantics***

Consistency + Persistency Model

# Formal Concurrency Models



# Formal Concurrency Models



# 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

```
// 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***

e.g. **`SFENCE`** on Intel

**`DSB`**$_{\texttt{full}}$ on ARM

# 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
```

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

# Explicit Persists

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

// recovery routine

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

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

***persist*** instructions

e.g. **CLWB**/**CLFLUSHOPT**/**CLFLUSH** on Intel (per cache line)

**DC-CVAP** on ARM (per cache line)

**psync** under epoch persistency (global)

# Here's Some Maths!

## ARM Persistence Semantics

- together w. ARM UK
- declarative specification
- discovered ***ambiguities in manual***

## Intel Persistence Semantics

- Together w. Intel US
- declarative specification
- operational specification
- equivalence theorem
- discovered ***ambiguities*** in manual

**Thread transitions:** $\textsc{Com} \xrightarrow{\textsc{Tid}:\textsc{Lab}_{\textsc{Px86}} \cup \{\epsilon\}} \textsc{Com}$

$$\frac{\mathsf{C}_1 \xrightarrow{\tau:l} \mathsf{C}_1'}{\textbf{let}\ \mathsf{a}{:=}\mathsf{C}_1\ \textbf{in}\ \mathsf{C}_2 \xrightarrow{\tau:l} \textbf{let}\ \mathsf{a}{:=}\mathsf{C}_1'\ \textbf{in}\ \mathsf{C}_2} \ (\textsc{T-Let1})$$

$$\frac{\mathsf{C} \xrightarrow{\tau:l} \mathsf{C}'}{\textbf{if}\ (\mathsf{C})\ \textbf{then}\ \mathsf{C}_1\ \textbf{else}\ \mathsf{C}_2 \xrightarrow{\tau:l} \textbf{if}\ (\mathsf{C}')\ \textbf{then}\ \mathsf{C}_1\ \textbf{else}\ \mathsf{C}_2} \ (\textsc{T-If1})$$

$$\frac{}{\textbf{repeat}\ \mathsf{C} \xrightarrow{\tau:\epsilon} \textbf{if}\ (\mathsf{C})\ \textbf{then}\ (\textbf{repeat}\ \mathsf{C})\ \textbf{else}\ 0} \ (\textsc{T-Repeat})$$

$$\frac{}{\textbf{load}(x) \xrightarrow{\tau:(\mathsf{R},x,v)} v} \ (\textsc{T-Read}) \qquad \frac{}{\textbf{FAA}(x,v) \xrightarrow{\tau:(\mathsf{U},x,v_0,v_0+v)} v_0} \ (\textsc{T-FAA})$$

$$\frac{v \neq v_1}{\textbf{CAS}(x,v_1,v_2) \xrightarrow{\tau:(\mathsf{R},x,v)} 0} \ (\textsc{T-CAS0}) \qquad \frac{}{\textbf{CAS}(x,v_1,v_2) \xrightarrow{\tau:(\mathsf{U},x,v_1,v_2)} 1} \ (\textsc{T-CAS1})$$

$$\frac{}{\textbf{flush}_{\text{opt}}\ x \xrightarrow{\tau:(\mathsf{FO},x)} 1} \ (\textsc{T-FO}) \qquad \frac{}{\textbf{flush}\ x \xrightarrow{\tau:(\mathsf{FL},x)} 1} \ (\textsc{T-FL})$$

**Program transitions:** $\textsc{Prog} \xrightarrow{\textsc{Tid}:\textsc{Lab}_{\textsc{Px86}} \cup \{\epsilon\}} \textsc{Prog}$

$$\frac{}{\textbf{let}\ \mathsf{a}{:=}v\ \textbf{in}\ \mathsf{C} \xrightarrow{\tau:\epsilon} \mathsf{C}[v/\mathsf{a}]} \ (\textsc{T-Let2})$$

$$\frac{v{\neq}0 \Rightarrow \mathsf{C}{=}\mathsf{C}_1 \quad v{=}0 \Rightarrow \mathsf{C}{=}\mathsf{C}_2}{\textbf{if}\ (v)\ \textbf{then}\ \mathsf{C}_1\ \textbf{else}\ \mathsf{C}_2 \xrightarrow{\tau:\epsilon} \mathsf{C}} \ (\textsc{T-If2})$$

$$\frac{}{\textbf{store}(x,v) \xrightarrow{\tau:(\mathsf{W},x,v)} v} \ (\textsc{T-Write})$$

$$\frac{}{\textbf{mfence} \xrightarrow{\tau:\mathsf{MF}} 1} \ (\textsc{T-MF}) \qquad \frac{}{\textbf{sfence} \xrightarrow{\tau:\mathsf{SF}} 1} \ (\textsc{T-SF})$$

$$\frac{\mathsf{P}(\tau) \xrightarrow{\tau:l} \mathsf{C}}{\mathsf{P} \xrightarrow{\tau:l} \mathsf{P}[\tau \mapsto \mathsf{C}]} \ (\textsc{Prog})$$

**Storage transitions:** $\textsc{Mem} \times \textsc{PBuff} \times \textsc{BMap} \xrightarrow{\textsc{Tid}:\textsc{Lab}_{\textsc{Px86}} \cup \{\epsilon\}} \textsc{Mem} \times \textsc{PBuff} \times \textsc{BMap}$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b} \quad o \in \{\mathsf{pfo},\mathsf{pfl}\} \quad x \in X \quad \langle \mathsf{sf} \rangle \notin \mathsf{b} \quad o{=}\mathsf{pfo} \Rightarrow \forall v.\forall x'{\in}X.\langle x',v\rangle,\langle \mathsf{fo},x'\rangle,\langle \mathsf{fl},x'\rangle \notin \mathsf{b} \quad o{=}\mathsf{pfl} \Rightarrow \forall y,v.\forall x'{\in}X.\langle y,v\rangle,\langle \mathsf{fo},x'\rangle,\langle \mathsf{fl},y\rangle \notin \mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB}.\langle \mathsf{per},x\rangle,\mathsf{B}[\tau \mapsto \mathsf{b}.\langle o,x\rangle]} \ (\textsc{M-BFetch})$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b}_1.\langle o,x\rangle.\mathsf{b}_2 \quad o{\in}\{\mathsf{pfo},\mathsf{pfl}\}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}_1.\mathsf{b}_2]} \ (\textsc{M-BDrop}) \qquad \frac{\mathsf{PB}{=}\mathsf{PB}_1.\langle x,v\rangle.\mathsf{PB}_2 \quad \forall y,v'.\langle x,v'\rangle,\langle \mathsf{per},y\rangle \notin \mathsf{PB}_1}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M}[x \mapsto v],\mathsf{PB}_1.\mathsf{PB}_2,\mathsf{B}} \ (\textsc{M-PropW})$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b} \quad x{\in}X \quad \forall y.\forall x'{\in}X.\langle \mathsf{pfl},y\rangle,\langle \mathsf{pfo},x'\rangle \notin \mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{W},x,v)} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}.\langle x,v\rangle]} \ (\textsc{M-Write}) \qquad \frac{\mathsf{B}(\tau){=}\mathsf{b} \quad \mathsf{rd}(\mathsf{M},\mathsf{PB},\mathsf{b},x){=}v}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{R},x,v)} \mathsf{M},\mathsf{PB},\mathsf{B}} \ (\textsc{M-Read})$$

$$\frac{\mathsf{B}(\tau){=}\epsilon \quad \mathsf{rd}(\mathsf{M},\mathsf{PB},\epsilon,x){=}v_r}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{U},x,v_r,v_w)} \mathsf{M},\mathsf{PB}[x \mapsto \mathsf{PB}(x).v_w],\mathsf{B}} \ (\textsc{M-RMW}) \qquad \frac{\mathsf{B}(\tau){=}\epsilon}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\mathsf{MF}} \mathsf{M},\mathsf{PB},\mathsf{B}} \ (\textsc{M-MF})$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b} \quad \forall x.\ \forall o \in \{\mathsf{pfo},\mathsf{pfl}\}.\ \langle o,x\rangle \notin \mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\mathsf{SF}} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}.\langle \mathsf{sf}\rangle]} \ (\textsc{M-SF}) \qquad \frac{\mathsf{B}(\tau){=}\langle \mathsf{sf}\rangle.\mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}]} \ (\textsc{M-BPropSF})$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b} \quad x \in X \quad \forall x' \in X.\langle \mathsf{pfl},x'\rangle,\langle \mathsf{pfo},x'\rangle \notin \mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{FO},x)} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}.\langle \mathsf{fo},x\rangle]} \ (\textsc{M-FO1}) \qquad \frac{\mathsf{B}(\tau){=}\mathsf{b}_1.\langle \mathsf{pfo},x\rangle.\mathsf{b}_2 \quad x{\in}X \quad \forall x'{\in}X.\ \langle \mathsf{pfo},x'\rangle,\langle \mathsf{pfl},x'\rangle \notin \mathsf{b}_1}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{FO},x)} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}_1.\mathsf{b}_2]}$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b} \quad x \in X \quad \forall y.\forall x' \in X.\langle \mathsf{pfl},y\rangle,\langle \mathsf{pfo},x'\rangle \notin \mathsf{b}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{FL},x)} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}.\langle \mathsf{pfl},x\rangle]} \ (\textsc{M-FL1}) \qquad \frac{\mathsf{B}(\tau){=}\mathsf{b}_1.\langle \mathsf{pfl},x\rangle.\mathsf{b}_2 \quad x{\in}X \quad \forall x'{\in}X.\ \langle \mathsf{pfo},x'\rangle \notin \mathsf{b}_1 \quad \forall y.}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:(\mathsf{FL},x)} \mathsf{M},\mathsf{PB},\mathsf{B}[\tau \mapsto \mathsf{b}_1.\mathsf{b}_2]}$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b}_1.\langle x,v\rangle.\mathsf{b}_2 \quad x{\in}X \quad \forall y,v.\forall x'{\in}X.\langle \mathsf{sf}\rangle,\langle y,v\rangle,\langle \mathsf{fl},y\rangle,\langle \mathsf{fo},x'\rangle \notin \mathsf{b}_1}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB}.\langle x,v\rangle,\mathsf{B}[\tau \mapsto \mathsf{b}_1.\mathsf{b}_2]} \ (\textsc{M-BPropW}) \qquad \frac{\mathsf{PB}{=}\mathsf{PB}_1.\langle \mathsf{per},x\rangle.\mathsf{PB}_2 \quad \forall y,v.\langle x,v\rangle,\langle \mathsf{p}}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB}_1.\mathsf{PB}_2,\mathsf{B}}$$

$$\frac{\mathsf{B}(\tau){=}\mathsf{b}_1.\langle o,x\rangle.\mathsf{b}_2 \quad o{\in}\{\mathsf{fo},\mathsf{fl}\} \quad x{\in}X \quad \langle \mathsf{sf}\rangle \notin \mathsf{b}_1 \quad o{=}\mathsf{fo} \Rightarrow \forall v.\forall x'{\in}X.\ \langle x',v\rangle,\langle \mathsf{fo},x'\rangle,\langle \mathsf{fl},x'\rangle \notin \mathsf{b}_1 \quad o{=}\mathsf{fl} \Rightarrow \forall y,v.\forall x'{\in}X.\langle y,v\rangle,\langle \mathsf{fo},x'\rangle,\langle \mathsf{fl},y\rangle \notin \mathsf{b}_1}{\mathsf{M},\mathsf{PB},\mathsf{B} \xrightarrow{\tau:\epsilon} \mathsf{M},\mathsf{PB}.\langle \mathsf{per},x\rangle,\mathsf{B}[\tau \mapsto \mathsf{b}_1.\mathsf{b}_2]} \ (\textsc{M-BPropP})$$

# Here's Some Maths!

ARM Persis

- together w
- declarative
- discoverec

Intel Persist

- Together w
- declarative
- operationa
- equivalenc
- discoverec

## ***Problem***

***counter-intuitive*** semantics
***low-level*** hardware details

## ***Solution***

***high-level, hardware-agnostic***
NVM libraries:

***Persistent Transactions***

# Transactions: NVM vs. TM/Databases

- ***TM*** (Transactional Memory): *run on **volatile** hardware*
  - ➡ ***No*** persistency guarantees

- ***Database*** transactions*: run on **persistent** hardware*
  - ➡ ***Specific*** persistency guarantees
    - ***only strict*** persistency
    - ***only synchronous*** persistency

- ***NVM*** transactions: run on ***persistent*** hardware
  - ➡ ***Range*** of persistency guarantees
    - ***strict or relaxed*** persistency
    - ***synchronous or asynchronous*** persistency

# 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:

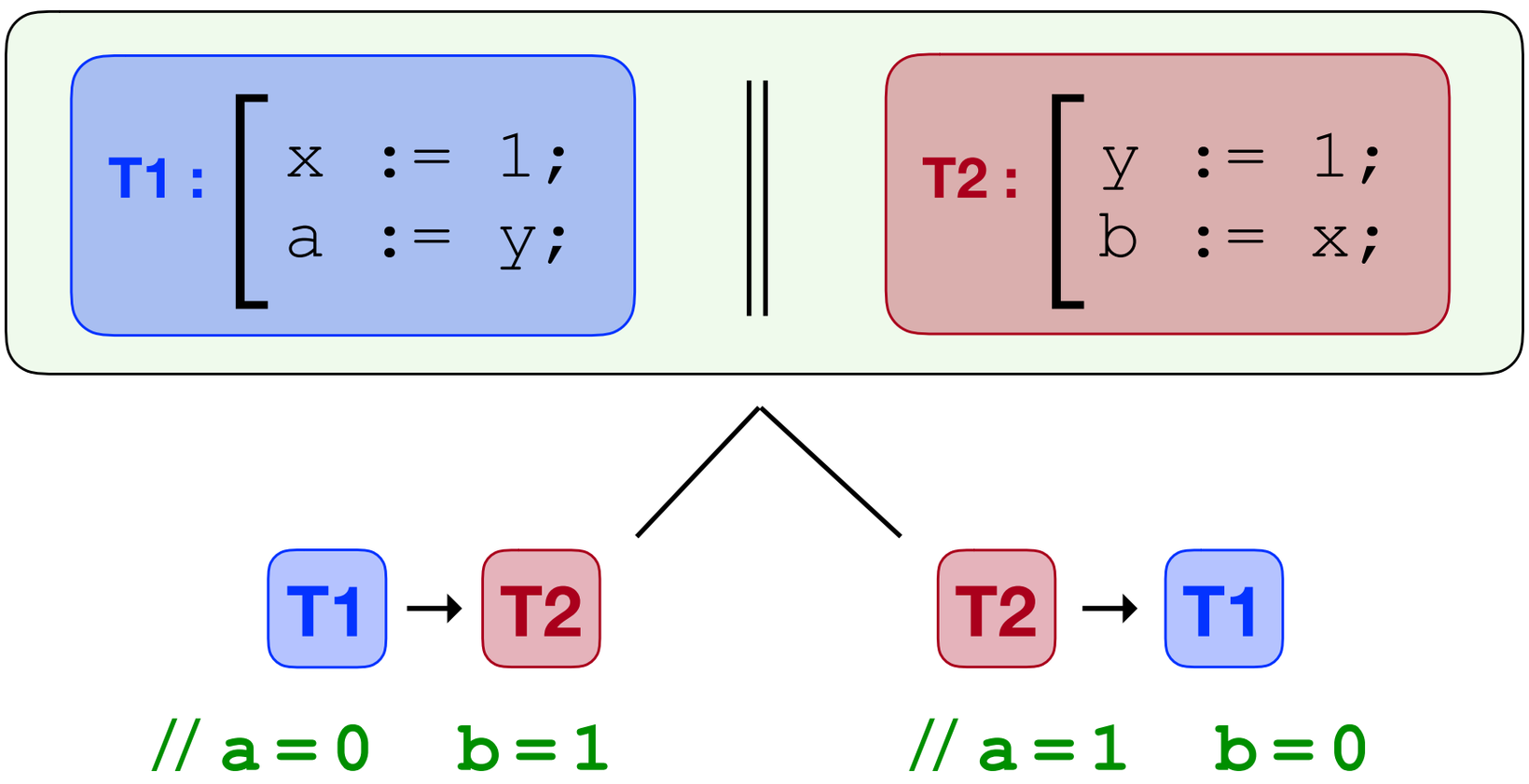
- ***atomic*** work unit:
  - ➡ all-or-nothing writes
  - ➡ all-or-nothing ***persists***
- ***consistent*** (e.g. serialisable)
- ***persistent*** (e.g. ***persistently serialisable***)

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

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

# Serialisability (SER)

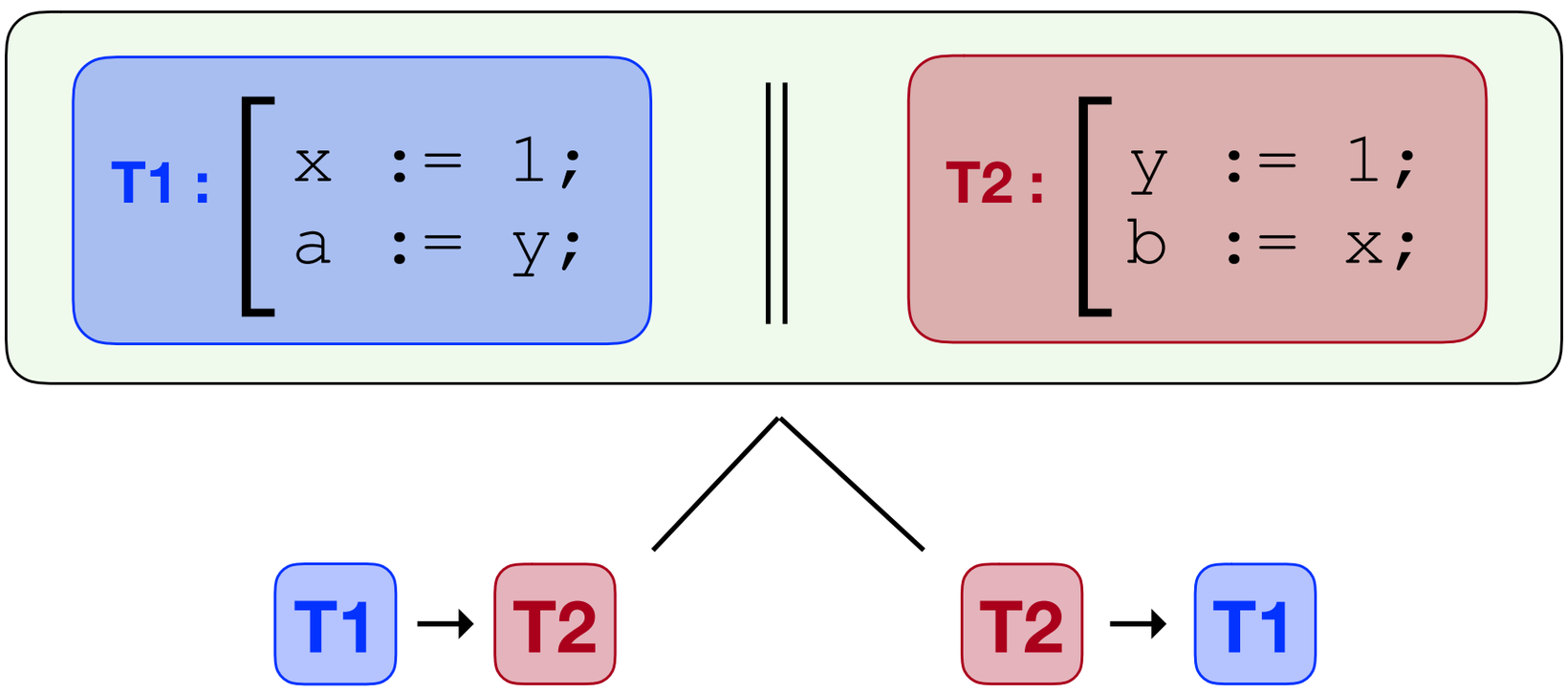
***All*** transactions appear to ***execute*** in a sequential order



# ***Persistent*** Serialisability (PSER)

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

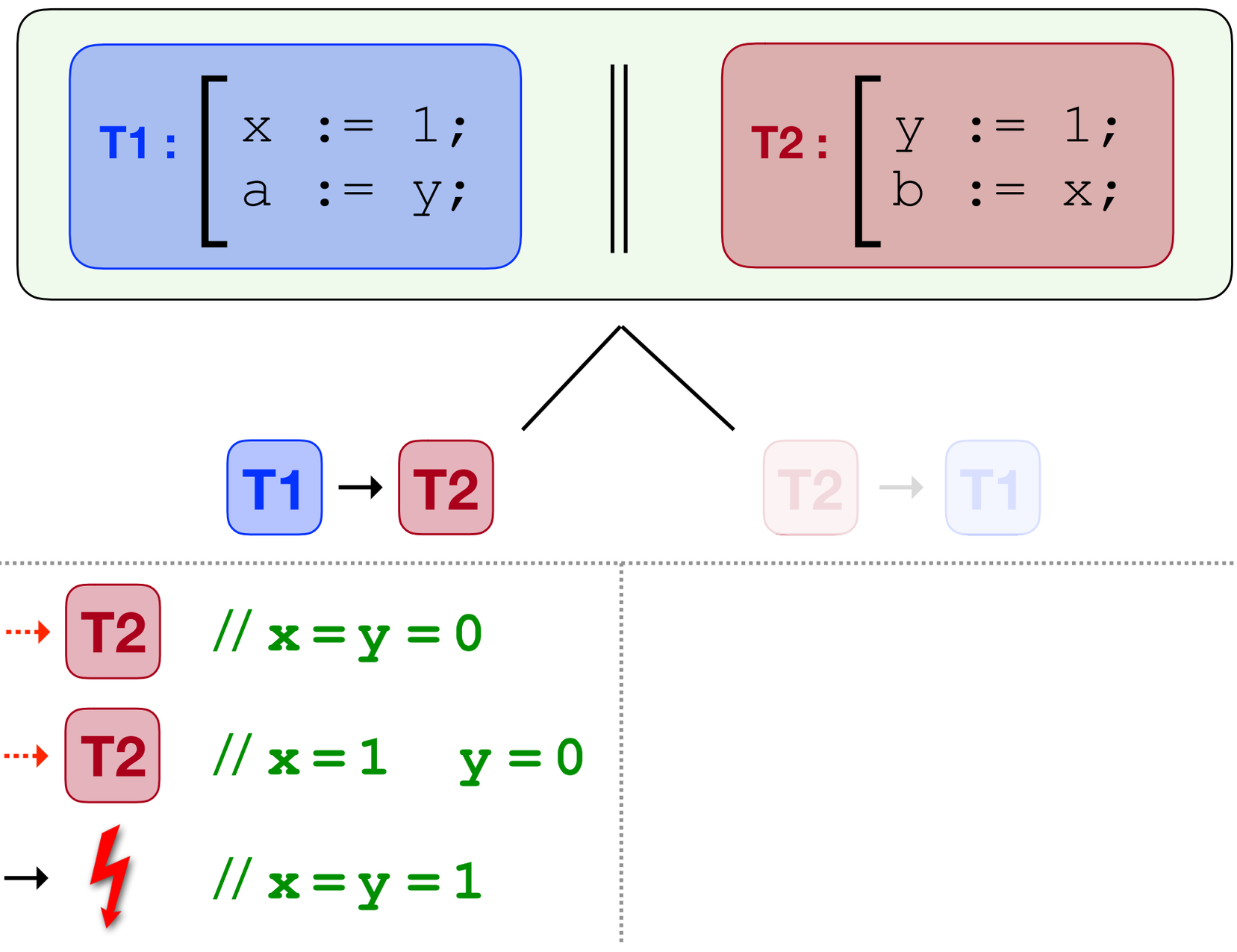
***A prefix*** of transactions appears to ***persist*** in the ***same sequential order***



# ***Persistent*** Serialisability (PSER)

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

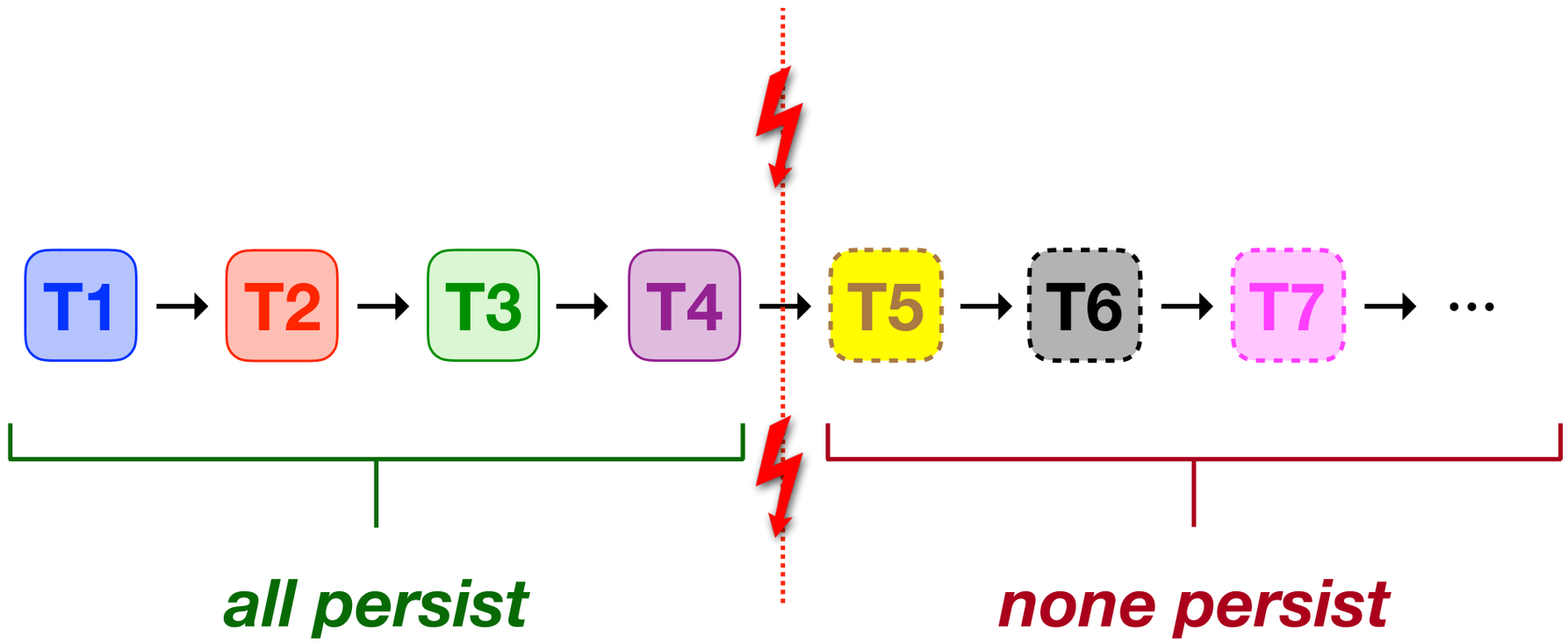
***A prefix*** of transactions appears to ***persist*** in the ***same sequential order***



# ***Persistent*** Serialisability (PSER)

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

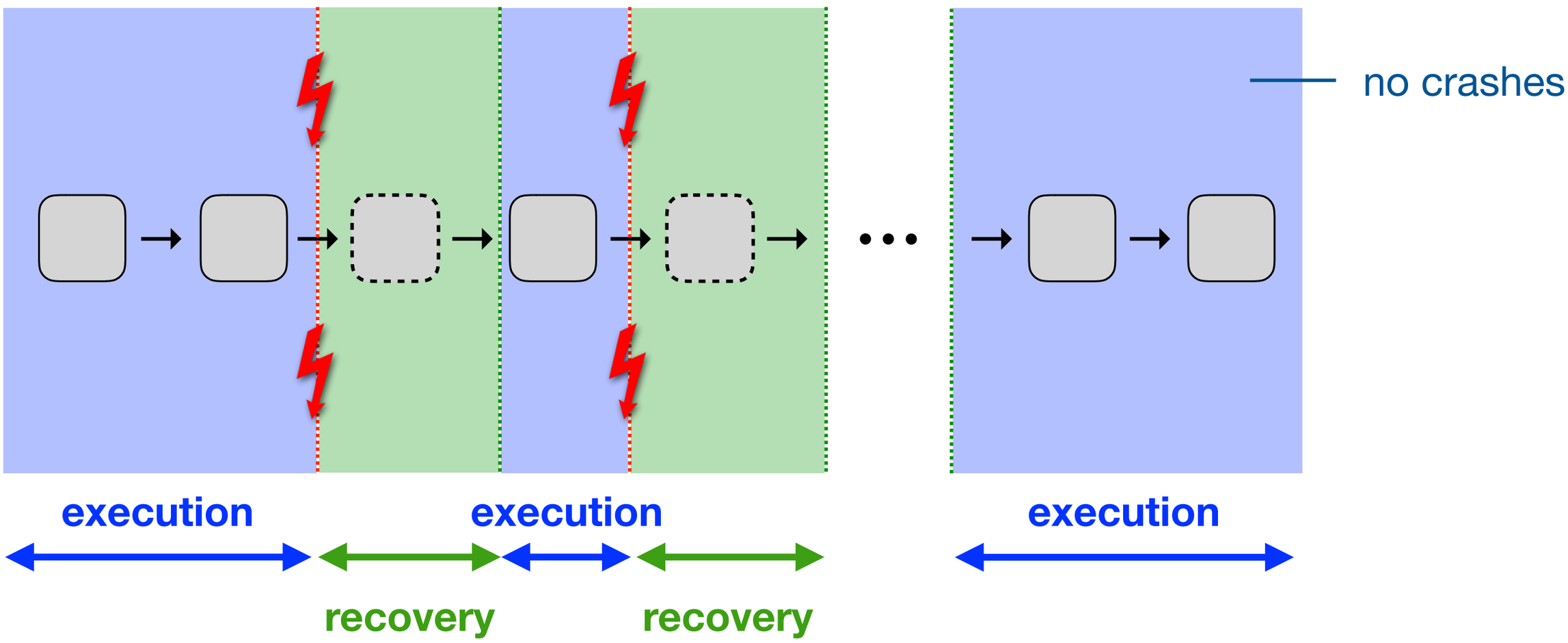
***A prefix*** of transactions appears to ***persist*** in the ***same sequential order***



# ***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***

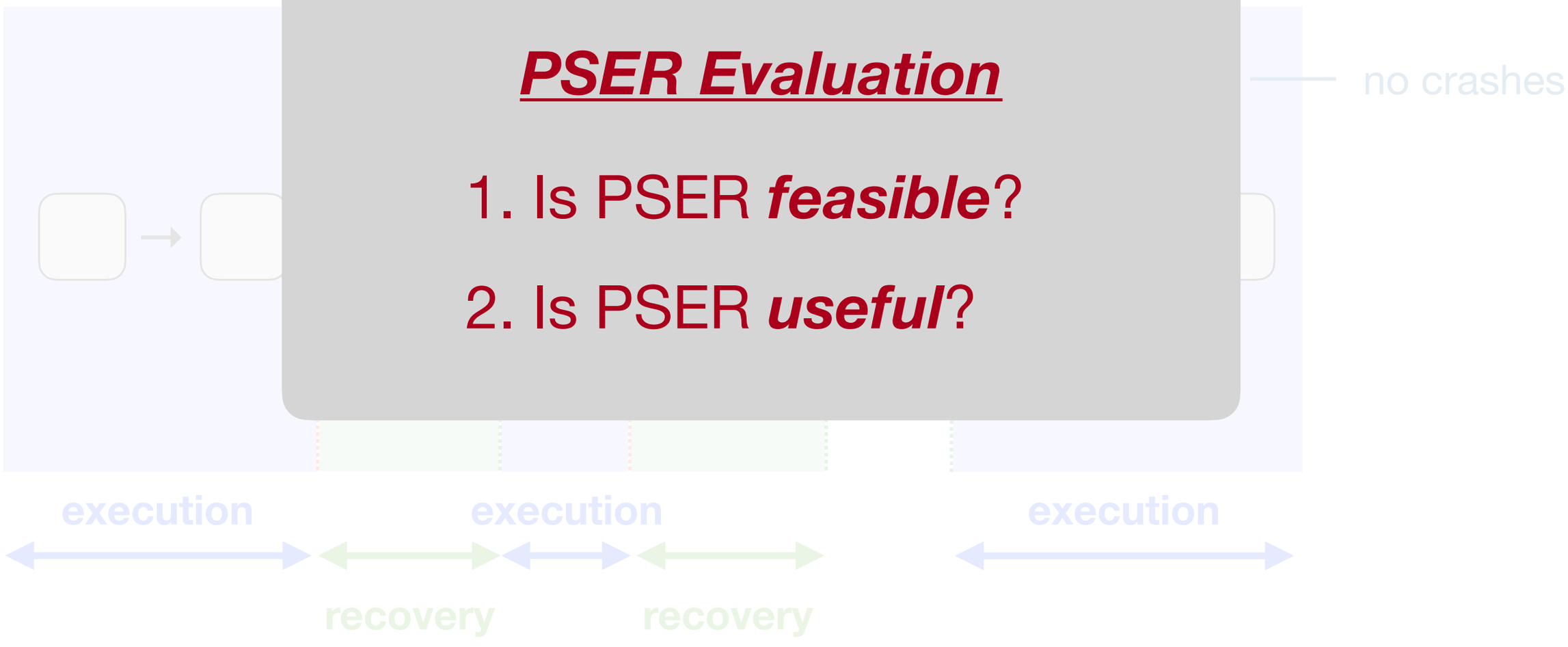
***Strong*** guarantees

***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***




***PSER Evaluation***

1. Is PSER ***feasible***?

2. Is PSER ***useful***?

# Is PSER ***Feasible***?

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

✓ PSER ***implementation*** in ***Intel***

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

- add code for ***persistence*** — e.g. **`psync`**
- 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***

✓ PSER ***implementation*** in ***Intel***

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

***Yes!***

***Correct Implementation*** in ***ARM*** & ***Intel***

⇒ 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*
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 >
```

```
  < deq_body > }
```

sequential queue imp.

***correct***
***concurrent*** & ***persistent***
queue imp.

# Summary

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

- ***ARM***
- ***Intel***

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

- ***PSER***
- ***Feasibility***: implemented PSER on ARM and Intel
- ***Utility***: PSER for concurrent & persistent library implementation

? Future Work:

- other transactional models
- model checking algorithms
- program logics

Thank You for Listening!

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