AARHUS UNIVERSITY

Spirea

A Concurrent Separation Logic For Weak Persistent Memory

Simon Friis Vindum

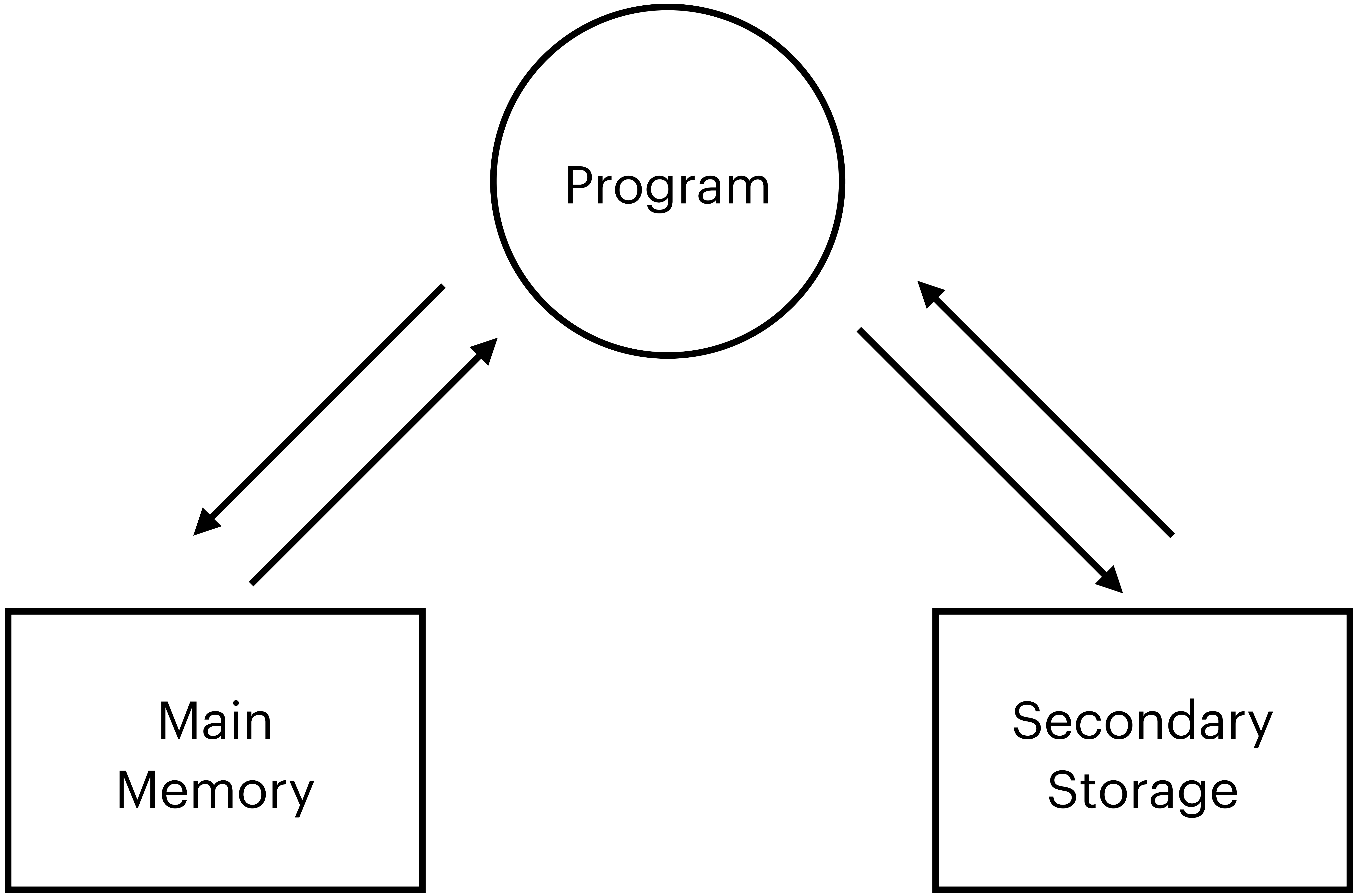
Lars Birkedal

DRAM

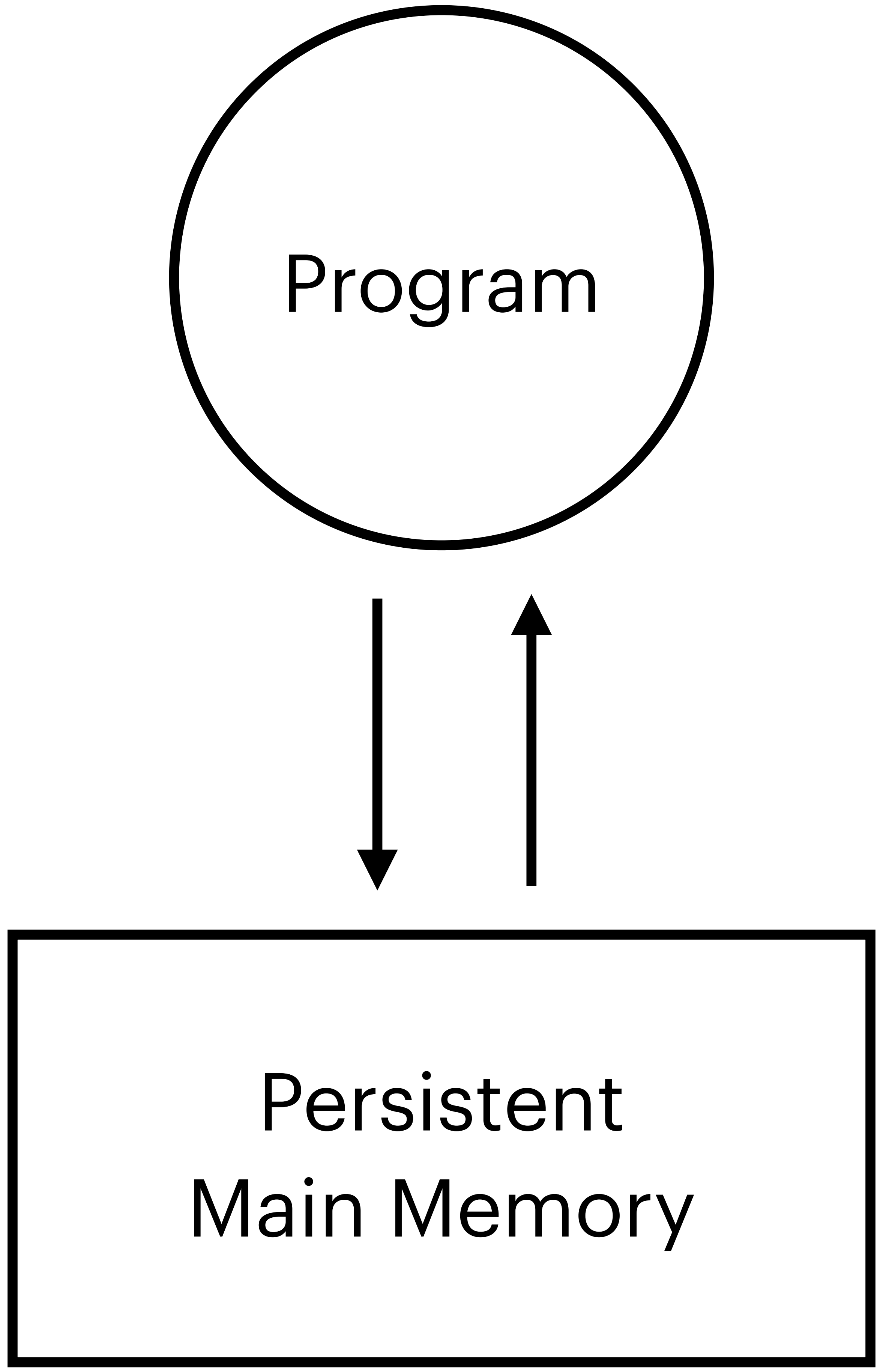
- Has been used to implement main memory since the 1970s
- Two big problems:
  - DRAM density is not expected to increase going forward
  - DRAM is power hungry – accounts for 25% of the power usage in data centers

New memory technologies are
non-volatile/persistent

Programming for *volatile* memory



Programming for *persistent* memory



Stuff build for persistent memory

- Researches and companies have been prolific building things for NVM.
  - Durable data-structures
    - Trees, hash-tables, queues, etc.
  - Key-value stores
  - Memory allocators
  - Garbage collectors
  - Transactions
  - And more …

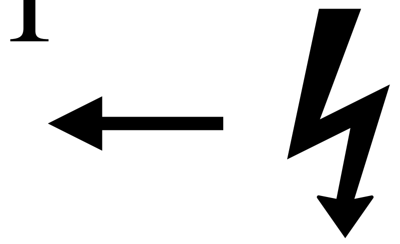
Challenges

- Since persistent memory is durable storage programs have to worry about crashes. I.e. be crash-safe.
- Writes to persistent memory are *asynchronous* and *weakly ordered.*

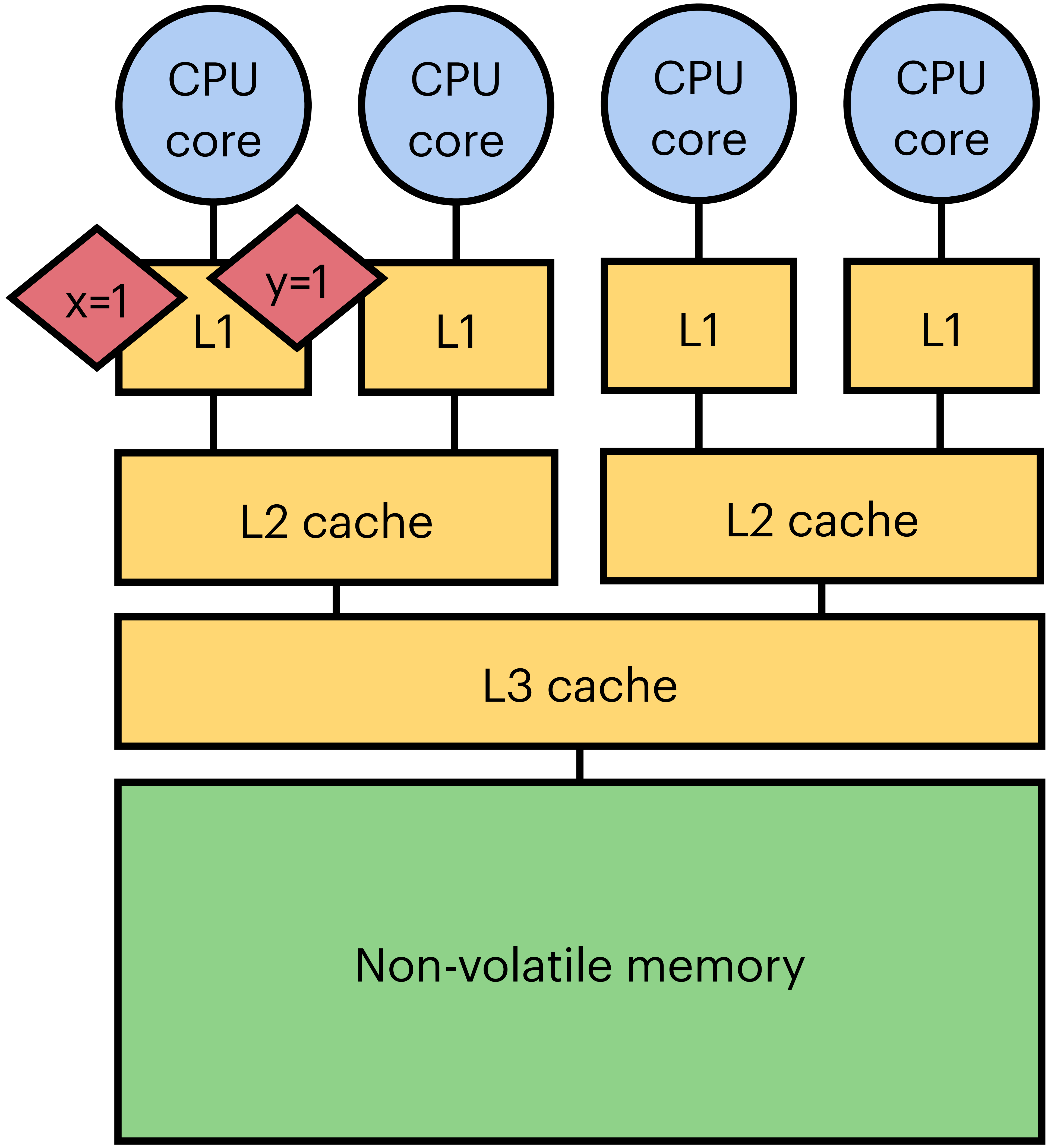
Initial Memory: $x = 0 \land y = 0$

$x \leftarrow 1$

$y \leftarrow 1$

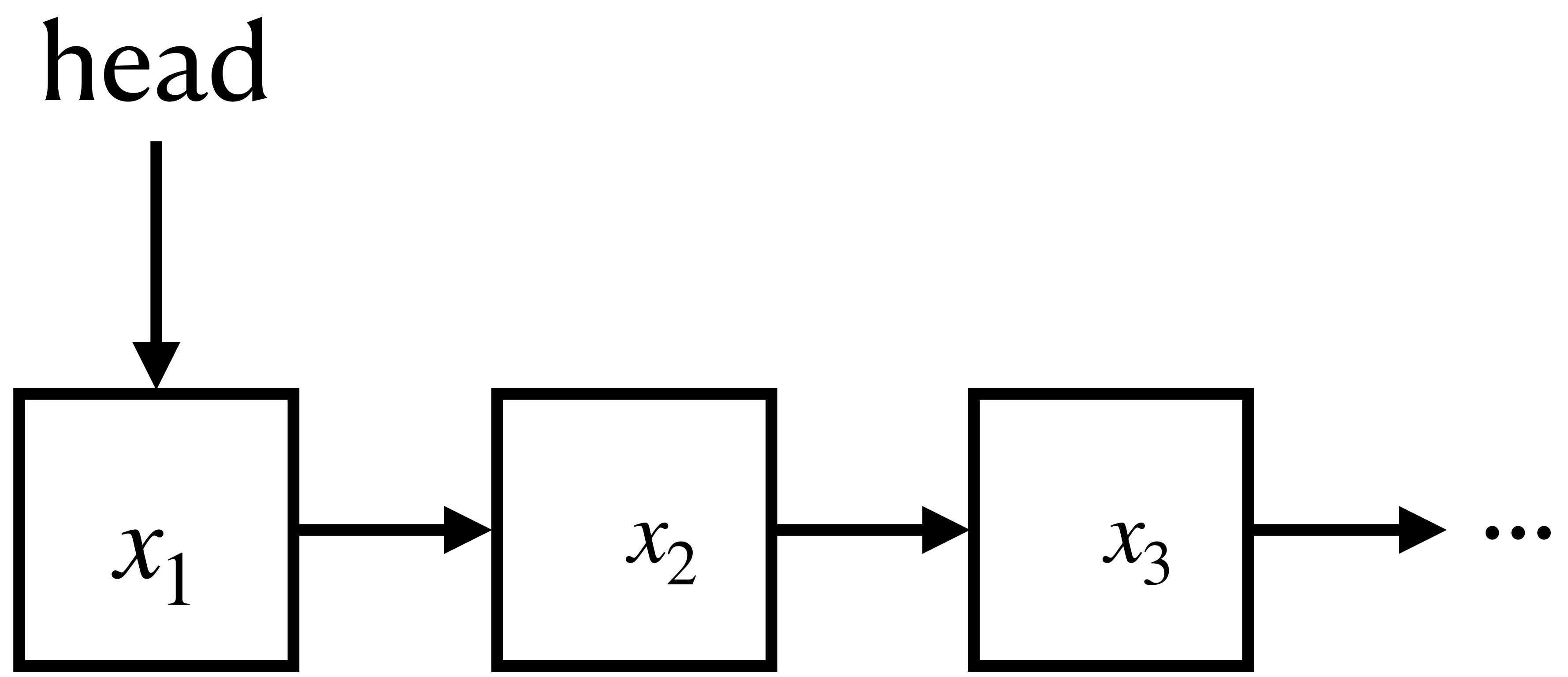


After Crash: $x = 0 \land y = 1$ is possible.



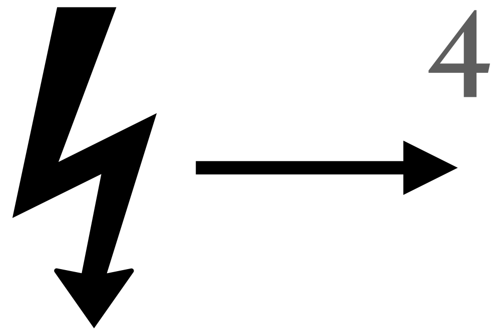
push(val, head)

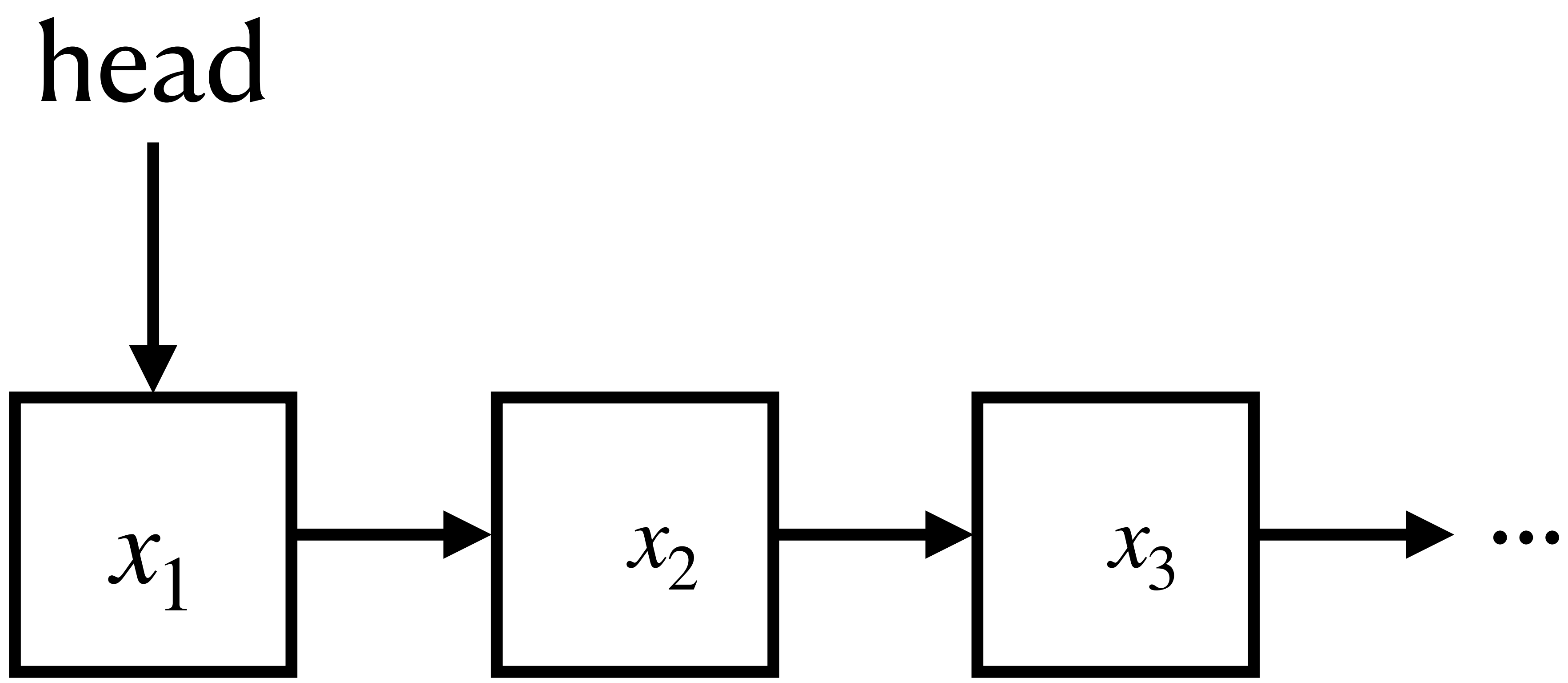
1. oldHead = ! head
2. node = allocateNode(val)
3. node.next ← oldHead
4. head ← node



push(val, head)

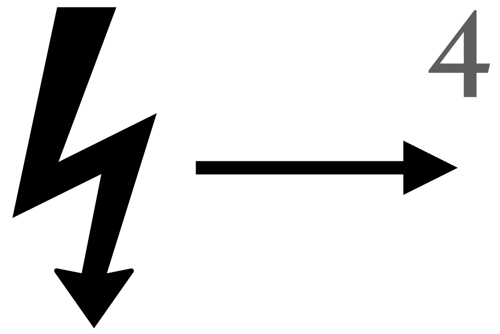
1. oldHead = ! head
2. node = allocateNode(val)
3. node.next ← oldHead
4. head ← node

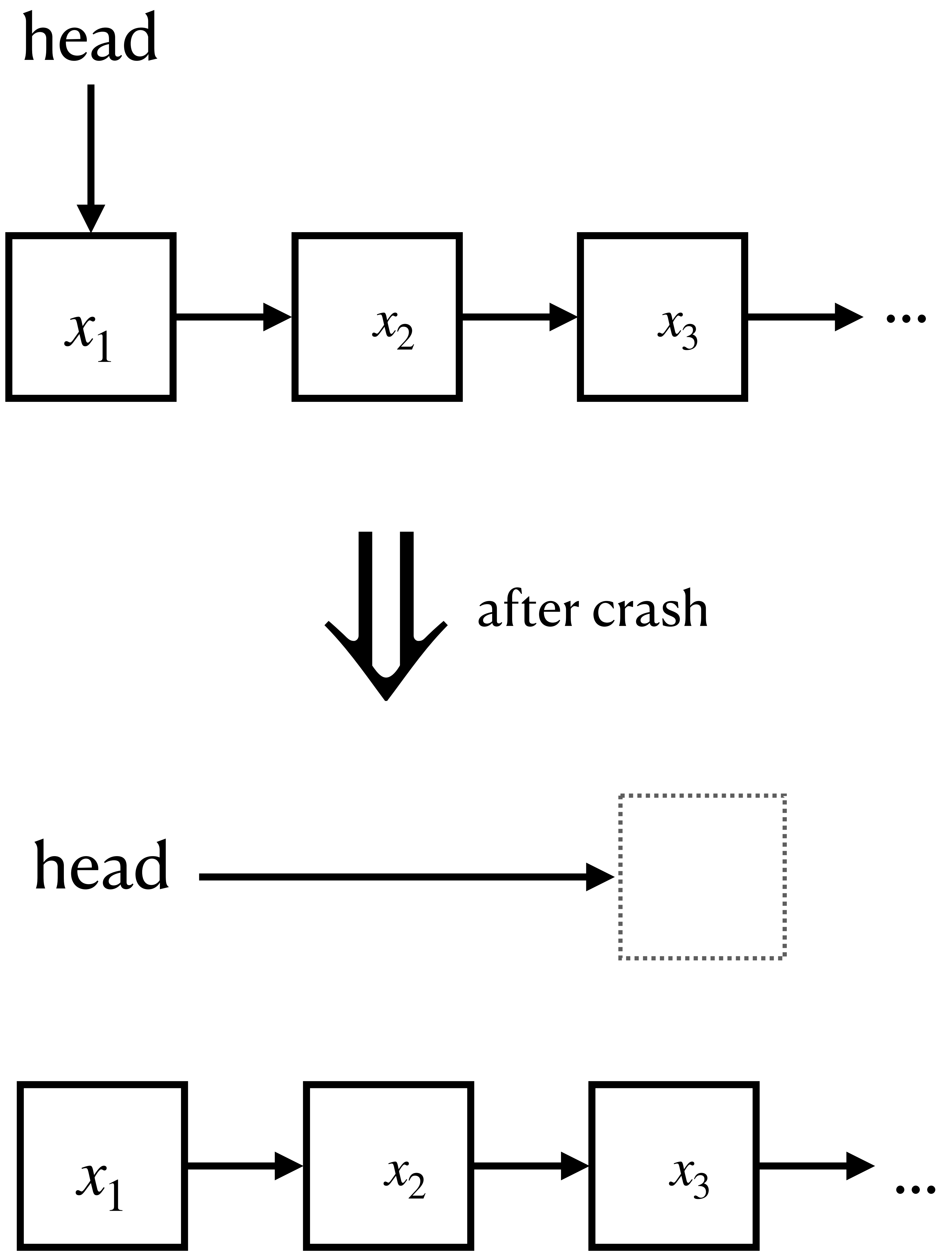




push(val, head)

1. oldHead = ! head
2. node = allocateNode(val)
3. node.next ← oldHead
4. head ← node





```
push(val, head)
    oldHead = ! head
    node = makeNode(val)
    node.next ← oldHead
    flush node
    flush node.next
    fence
    head ← node
```

Our Goal

- To create a program logic that can verify programs that use weak persistent memory.

Our Work

- A **small-step operational semantics** for the *explicit-epoch persistency model*
- Instantiated Iris/Perennial to arrive at a low-level logic
- Built the higher level **Spirea** logic on top of the low-level logic
- Verified examples
  - Tricky synthetic examples
  - Durable data structures
- Mechanized in Coq

At A Glance

Locations

$\boxed{\ell \mid \pi}$

$\ell \hookrightarrow_{\mathsf{NA}} \overrightarrow{\sigma}$

$\ell \hookrightarrow_{\mathsf{AT}} \overrightarrow{\sigma}$

Lower Bounds

$\ell \gtrsim_{\mathsf{p}} \sigma$

$\ell \gtrsim_{\mathsf{f}} \sigma$

$\ell \gtrsim_{\mathsf{s}} \sigma$

Crash Modalities

$\langle \mathsf{PC} \rangle\, P$

$\langle \mathsf{PCF} \rangle\, P$

$\langle \mathsf{ifRec} \rangle_{\ell}\, P$

View Modalities

$\langle \mathsf{obj} \rangle\, P$

$\langle \mathsf{NF} \rangle\, P$

$\langle \mathsf{NB} \rangle\, P$

Post Fence Modalities

$\langle \mathsf{PF} \rangle\, P$

$\langle \mathsf{PF}_{\mathsf{F}} \rangle\, P$

Weakest Precondition

$\mathrm{wpc}\ e\ \{Q\}\{Q_c\}$

$\mathrm{wpr}\ e \circlearrowleft e_r\ \{Q\}\{Q_c\}$

Modalities For Crashes

$$\langle \mathsf{PC} \rangle \, P$$

$$\langle \mathsf{PCF} \rangle \, P$$

$$\langle \mathsf{ifRec} \rangle_{\ell} \, P$$

$$\mathrm{isStack}(\ell, \phi) -\!\!\star \langle \mathsf{PC} \rangle\, \mathrm{isStack}(\ell, \phi)$$

$$\mathrm{isStack}(\ell, \phi) -\!\!\star \langle \mathsf{PC} \rangle\, \langle \mathsf{ifRec} \rangle_{\ell}\, \mathrm{isStack}(\ell, \phi)$$

Normal Safety

$$\langle e, \sigma \rangle \to \cdots \to \langle v, \sigma_n \rangle$$

Crash-safety

$$\langle e, \sigma \rangle \to \cdots \to \langle e_i, \sigma_i \rangle \xrightarrow{\lightning} \langle e_r, \sigma_{i+1} \rangle \to \cdots \to$$

$$\vdots$$

$$\cdots \to \langle e_j, \sigma_j \rangle \xrightarrow{\lightning} \langle e_r, \sigma_{j+1} \rangle \to \cdots \to \langle v, \sigma_n \rangle$$

Crash Step

M-CRASH

$$\frac{\begin{array}{c}\text{consistent}(\sigma, \mathcal{P}, \mathcal{C}) \\ \mathsf{dom}(\sigma') = \mathsf{dom}(\mathcal{C}) \qquad \forall \ell \in \mathsf{dom}(\mathcal{C}).\, \sigma'(\ell) = \{0 \mapsto \langle \sigma(\ell)(\mathcal{C}(\ell)).\text{v}, \bot, \bot, \bot \rangle\} \\ \mathsf{dom}(\mathcal{P}') = \mathsf{dom}(\mathcal{C}) \qquad \forall \ell \in \mathsf{dom}(\mathcal{C}).\, \mathcal{P}'(\ell) = 0\end{array}}{\langle \sigma, \mathcal{P} \rangle \xrightarrow{\lightning} \langle \sigma', \mathcal{P}' \rangle}$$

memory configuration

Recovery Weakest Precondition (from Perennial)

$$\text{wpr}\ e \circlearrowleft e_r\ \{Q\}\{Q_c\}$$

post condition if no crashes

post condition if ≥ 1 crashes

Recovery Weakest Precondition

$$\frac{\mathsf{wpc}\ e\ \{Q\}\{Q_r\} \qquad Q_r \twoheadrightarrow \langle \mathsf{PC} \rangle\ \mathsf{wpc}\ e_r\ \{Q_c\}\{Q_r\}}{\mathsf{wpr}\ e \circlearrowleft e_r\ \{Q\}\{Q_c\}}$$

Crash-Aware Protocols

Crash-Aware Protocols

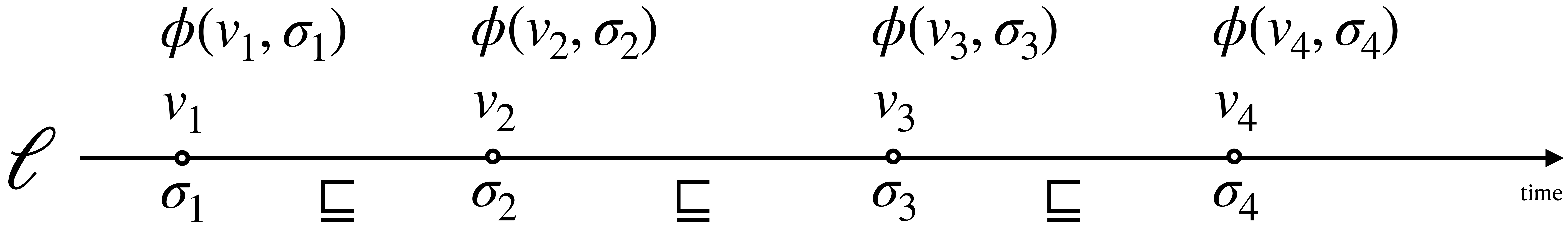
A *crash-aware protocol* $(\Sigma, \sqsubseteq, \phi, \psi)$ consists of a countable set of states Σ, a preorder $\sqsubseteq \in \sigma \times \sigma$ on the states, an invariant $\phi : \Sigma \times \mathrm{Val} \to \mathrm{dProp}$, and a function $\psi : \Sigma \to \Sigma$ that is monotone with respect to $\sqsubseteq$.

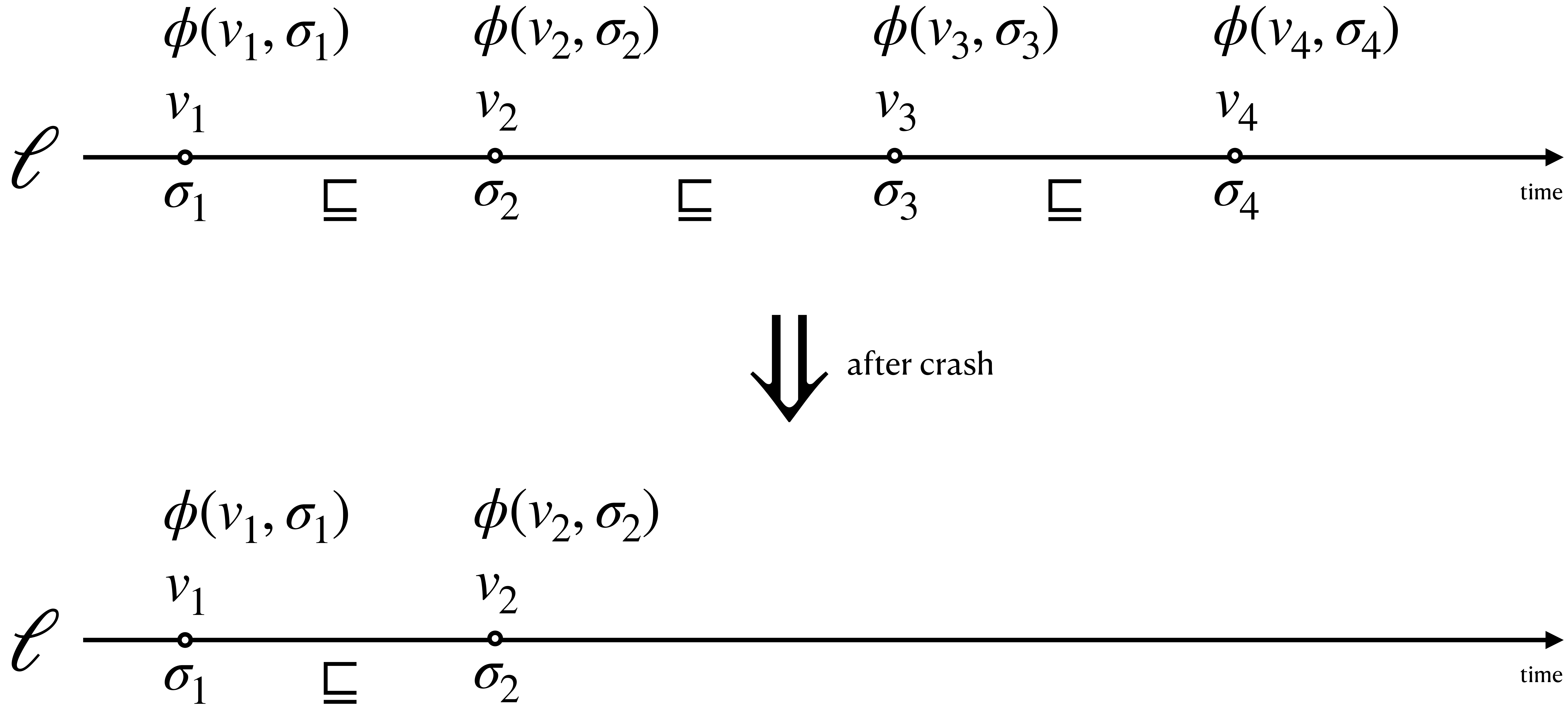
Additionally, the following two conditions must be satisfied:

1/ $\forall \sigma, v \,.\, \phi(\sigma, v) \vdash \langle \mathrm{NB} \rangle\, \phi(\sigma, v)$

2/ For all $\sigma \in \Sigma$ and $v \in \mathrm{Val}$ it is the case that $\phi(\sigma, v) \vdash \langle \mathrm{PCF} \rangle\, \phi(\psi(\sigma), v)$

- A protocol associates every location with
  - a set of states Σ and a preorder $\sqsubseteq$
  - a predicate $\phi : \Sigma \times \text{Val} \rightarrow \text{dProp}$




$\phi(v_1, \sigma_1)$
$\phi(v_2, \sigma_2)$
$\phi(v_3, \sigma_3)$
$\phi(v_4, \sigma_4)$
v_1
v_2
v_3
v_4
ℓ
$\sigma_1 \sqsubseteq \sigma_2 \sqsubseteq \sigma_3 \sqsubseteq \sigma_4$
time
after crash
$\phi(v_1, \sigma_1)$
$\phi(v_2, \sigma_2)$
v_1
v_2
ℓ
$\sigma_1 \sqsubseteq \sigma_2$
time

Points-to for non-atomics

$$\ell \hookrightarrow_{\mathsf{NA}} [\sigma_1, \sigma_2, \ldots, \sigma_n]$$

A post-crash modality rule

$$\ell \hookrightarrow_{\mathsf{NA}} [\sigma_1, \sigma_2, \ldots, \sigma_n] \vdash \langle \mathrm{PC} \rangle \, \langle \mathrm{ifRec} \rangle_\ell \, \exists i \leq n.\, \ell \hookrightarrow_{\mathsf{NA}} [\sigma_1, \sigma_2, \ldots, \sigma_i]$$

Location Lower Bounds

$\ell \gtrsim_{\mathsf{p}} \sigma$ $\ell \gtrsim_{\mathsf{f}} \sigma$ $\ell \gtrsim_{\mathsf{s}} \sigma$

Flush and fence

Initial Memory: $x = 0 \wedge y = 0$

$x \leftarrow 1$

flush x

fence

$y \leftarrow 1$

After Crash: If $y = 1$ then $x = 1$.

Post Fence Modalities

$$\langle \mathsf{PF} \rangle\, P$$

$$\langle \mathsf{PF_S} \rangle\, P$$

Rules for fence and flush

FLUSH

$$\{\ell \gtrsim_{\mathsf{s}} \sigma\}\ \textbf{flush}\ \ell\ \left\{().\ \langle \mathsf{PF} \rangle(\ell \gtrsim_{\mathsf{f}} \sigma) * \langle \mathsf{PF_S} \rangle(\ell \gtrsim_{\mathsf{p}} \sigma)\right\}$$

FENCE

$$\{\langle \mathsf{PF} \rangle\, P\}\ \textbf{fence}\ \{P\}$$

Non-Atomic Locations

NA-ALLOC

$$\{\phi(\sigma, v)\}\ \ \mathbf{ref}_{\mathsf{NA}}\ v\ \ \left\{\ell.\ \boxed{\ell \mid \pi} * \ell \hookrightarrow_{\mathsf{NA}} \sigma\right\}$$

NA-LOAD

$$\frac{\langle \mathrm{obj} \rangle\, \forall v.\, P * \phi(\vec{\sigma}_n, v) \twoheadrightarrow Q(v) * \phi(\vec{\sigma}_n, v)}{\left\{\boxed{\ell \mid \pi} * \ell \hookrightarrow_{\mathsf{NA}} \vec{\sigma} * P\right\}\ \ !_{\mathsf{NA}}\, \ell\ \ \{w.\, \ell \hookrightarrow_{\mathsf{NA}} \vec{\sigma} * Q(v)\}}$$

NA-STORE

$$\left\{\phi(\sigma, v) * \boxed{\ell \mid \pi} * \ell \hookrightarrow_{\mathsf{NA}} \vec{\sigma} * (\vec{\sigma})_n \sqsubseteq \sigma\right\}\ \ \ell \leftarrow_{\mathsf{NA}} v\ \ \{().\, \ell \hookrightarrow_{\mathsf{NA}} \vec{\sigma}\sigma\}$$