Weak-memory local reasoning

Ian Wehrman

September 26, 2012

Overview

Single-threaded program behavior w.r.t. an idealized computer model is complex.

Multi-threaded program behavior w.r.t. a **realistic** computer model is *really* complex.

Memory models

Specify interaction between programs and memory.

Description:

- Notion of state (an abstract representation of memory);
- Explanation of how values are read from/written in a given state.

Memory models

Different programs require different MMs:

Sequential imperative programs w/statically allocated memory:

$\text{State} \triangleq \text{Stack}$ where $\text{Stack} \triangleq \text{Variable} \rightharpoonup \text{Value}$

Sequential (or well-locked concurrent) imperative programs with dynamically allocated memory:

$\text{State} \triangleq \text{Stack} \times \text{Heap}$ where $\text{Heap} \triangleq \text{Address} \rightharpoonup \text{Value}$

Racy concurrent imperative programs:

$\text{State} \triangleq$ ... depends on the architecture.

Racy programs

Not all racy programs are broken:

e.g., lock-free concurrent data structures.

x86 MM

A weak, x86-like memory model:

$\mathrm{State} \triangleq \mathrm{Stack} \times \mathrm{Heap} \times \mathrm{WriteBufferArray} \times \mathrm{Lock}$

$\mathrm{WriteBufferArray} \triangleq \mathrm{Processor} \to \mathrm{WriteBuffer}$

$\mathrm{WriteBuffer} \triangleq \mathrm{Queue}[\mathrm{Write}]$

$\mathrm{Write} \triangleq \mathrm{Address} \times \mathrm{Value}$

$\mathrm{Lock} \triangleq \mathrm{Processor} + \perp$

x86 MM

On processor i:

store enqueues a new write on ith buffer;

load returns value of most recent write in ith buffer;
if none, then value in heap;

fence flushes all writes on ith buffer to the heap;

acquire (lock) or release (unlock) the global lock.

all but store block while j≠i holds lock.

Writes may commit nondeterministically!

Hoare logic

Program specifications: $\vdash \{ P \}\ c\ \{ Q \}$

command c is a sequential static-memory command;

precondition P describes an initial set of states;

postcondition Q describes a final set of states.

Meaning:

if c executes from a P-state, it terminates in a Q-state or diverges.

Separation logic

Extension of Hoare Logic: $\vdash \{ P \}\ c\ \{ Q \}$

enables sound reasoning about dynamic-memory commands;

additional assertions used to describe heap values;

all proved programs are memory-safe.

Concurrent separation logic

An extension of separation logic: $J \vdash \{ P \}\ c\ \{ Q \}$

c is a *concurrent* dynamic-memory command;

P and Q describe thread-private states;

invariant J describes environment-shared states;

all proved programs are well-locked and race-free.

(Required by simple memory model!)

Project

Goal: A program logic with an x86-like model.

Why?

Existing logics insufficient or unsound for racy programs.

Eventually wish to prove racy programs correct.

Explore concurrent reasoning in weak vs. strong MMs.

Project

Result: an x86-like variant of CSL.

Components:

1) a programming language;
2) an assertion logic;
3) a specification logic.

Project components

(1/3) Programming language:

C-like w/assignment, load, store, fence & locking primitives.

x86-like semantics.

NEW!

Project components

(2/3) Assertion logic:

Like the assertion language of SL/CSL, but more expressive. NEW!

Describe heaps **and** write buffers **and** the global lock.

x86-like semantics. NEW!

Ideally also a proof theory, but that's future work.

Project components

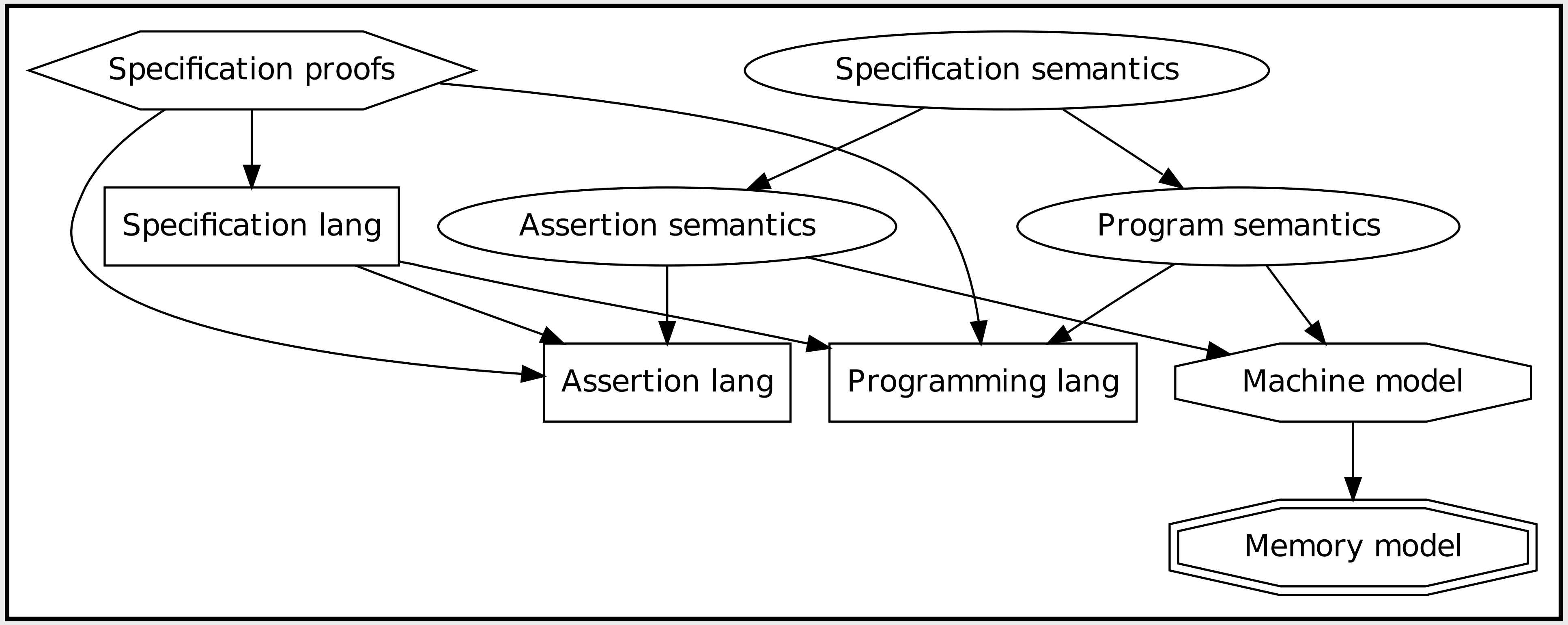
(3/3) Specification logic:

CSL-like specifications.

CSL-like proof theory, but with x86-specific adjustments. NEW!

x86-like semantics. NEW!

Project components



Agenda

Assertion language and models:

Language extends FOL and SL/CSL language;

New formulas for new state elements.

Design constraints from specification logic:

Expressive enough to describe x86 commands;

Constrained enough for sound, local reasoning.

Local reasoning

The **big idea** in SL/CSL:

Restrict reasoning to a small, relevant part of system state;

Then generalize to a complete description of system state.

Embodied by the **frame rule**:

$$\frac{J \vdash \{\, P \,\}\ c\ \{\, Q \,\}}{J \vdash \{\, R * P \,\}\ c\ \{\, R * Q \,\}}$$

Separation

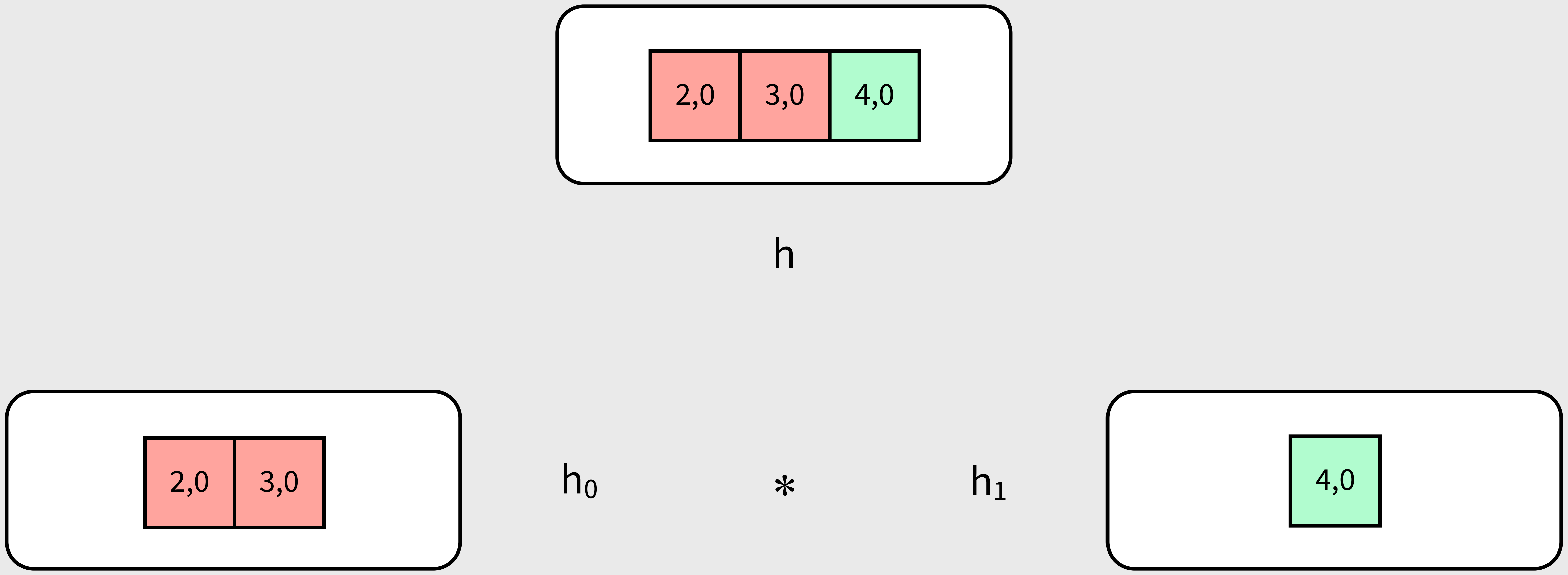
In SL/CSL:

$P * Q$ is the **separating conjunction** of assertions P and Q.

Describes heaps that can be partitioned into sub-heaps:

a sub-heap described by P and a sub-heap described by Q.

Separation



Spatial separation

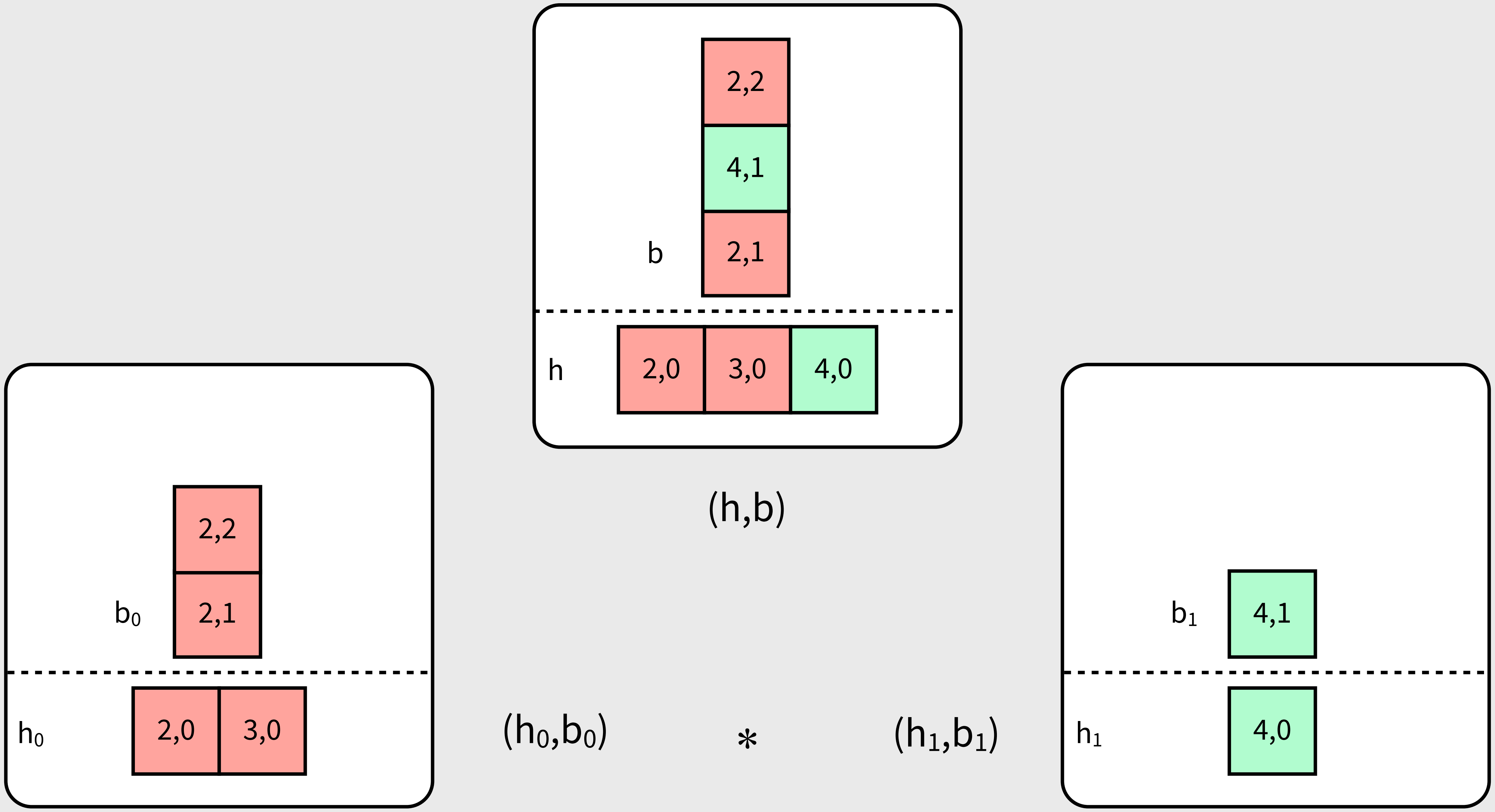
In x86-CSL:

$P * Q$ is called the **spatial separating conjunction** of P and Q.

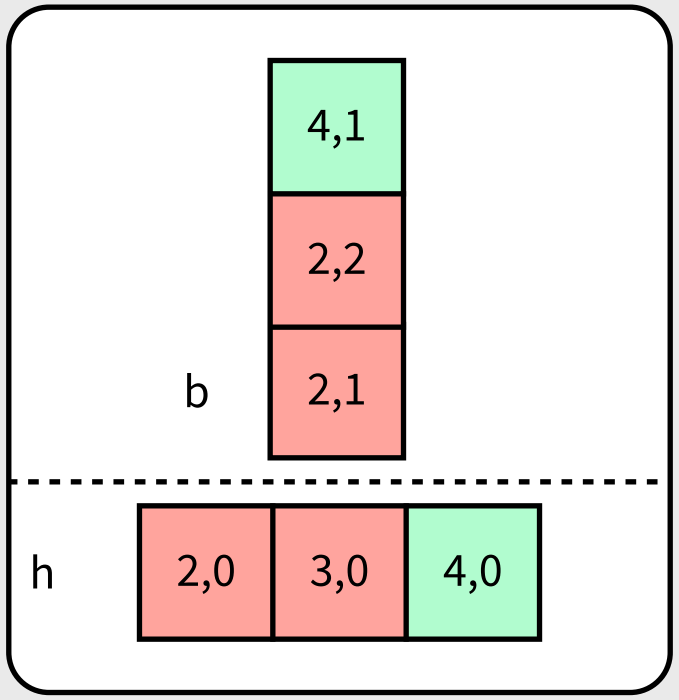
Describes x86 states that are separable by address:

a sub-state described by P and a sub-state described by Q.

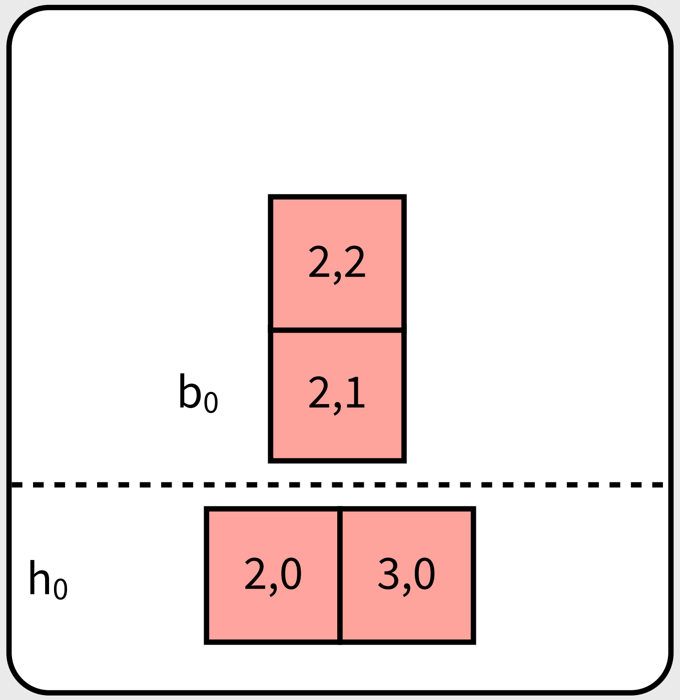
Spatial separation



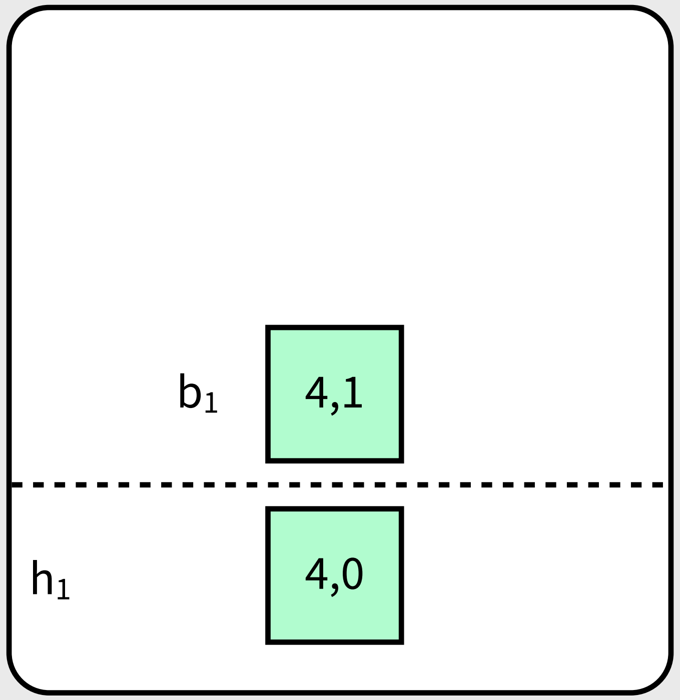
Spatial separation



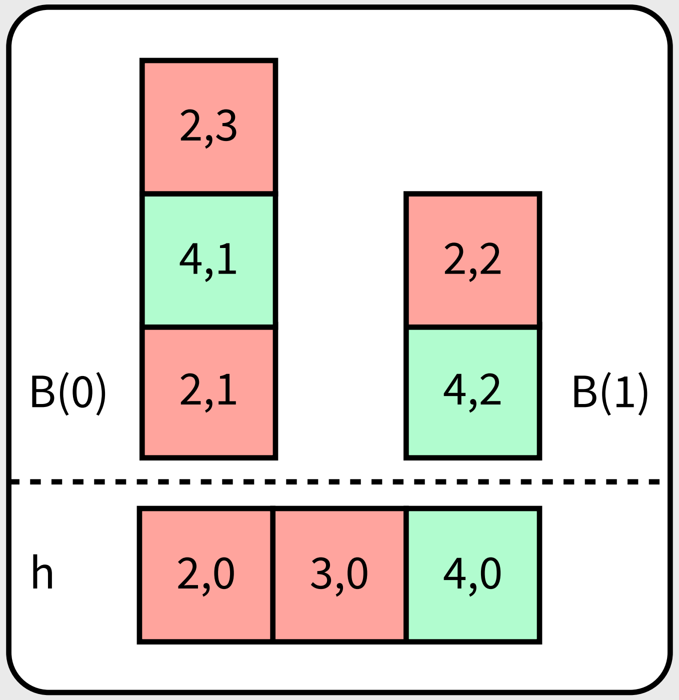
(h,b)



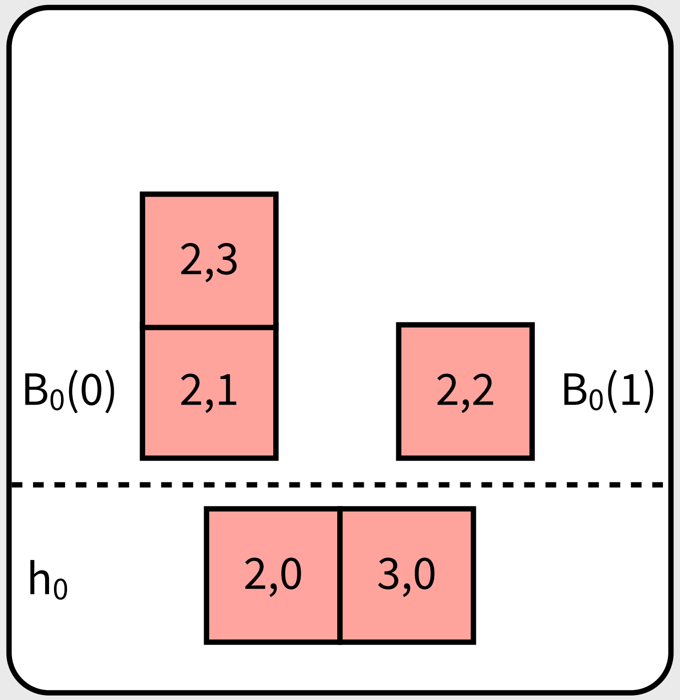
$(h_0,b_0) \quad * \quad (h_1,b_1)$



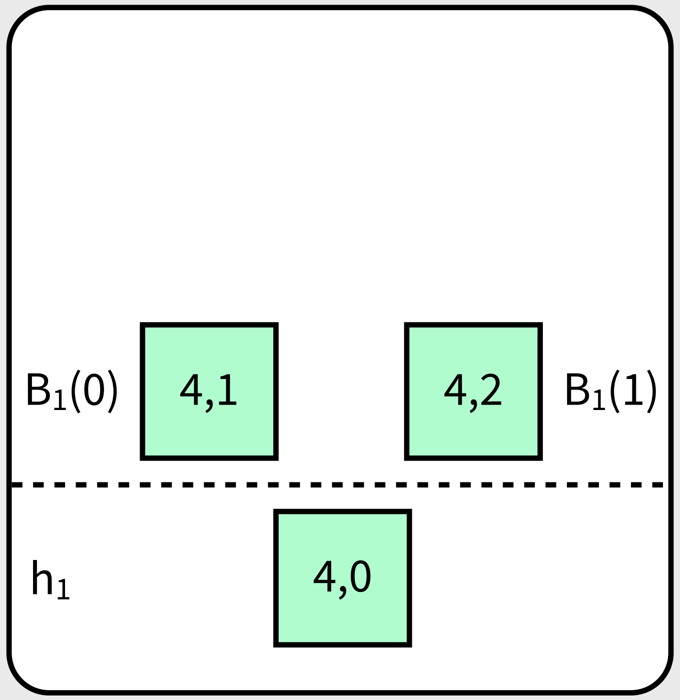
Spatial separation



(h,B)



(h_0,B_0) $*$ (h_1,B_1)

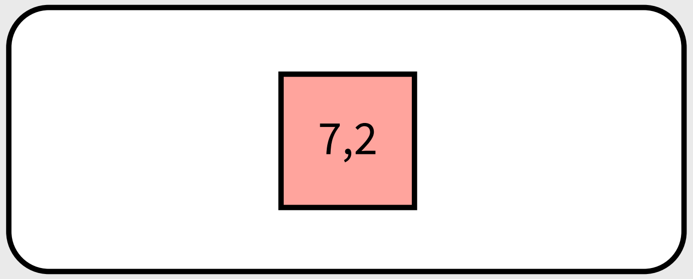


Heap values

In SL/CSL:

The **points-to assertion** describes a **single heap value**.

$7 \mapsto 2$

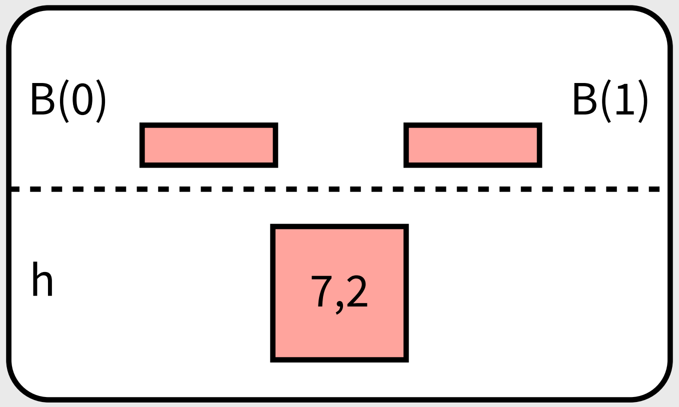


Heap values

In x86-CSL:

The points-to assertion describes a heap value and **empty buffers**.

$7 \mapsto 2$

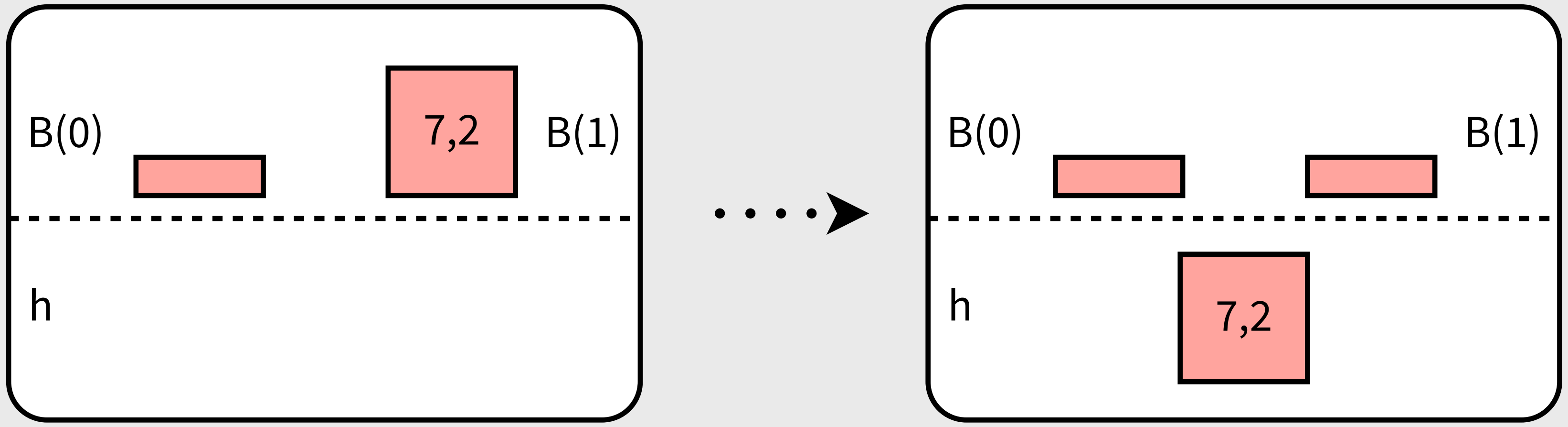


Buffered writes

In x86-CSL:

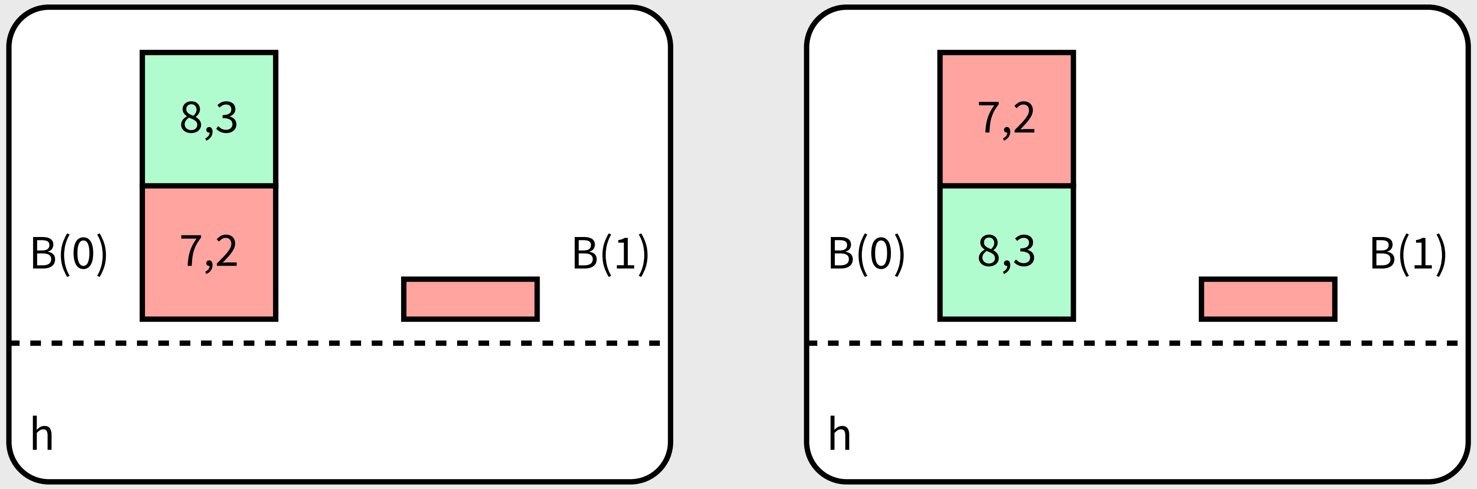
The **leads-to assertion** describes a **single buffered write**.

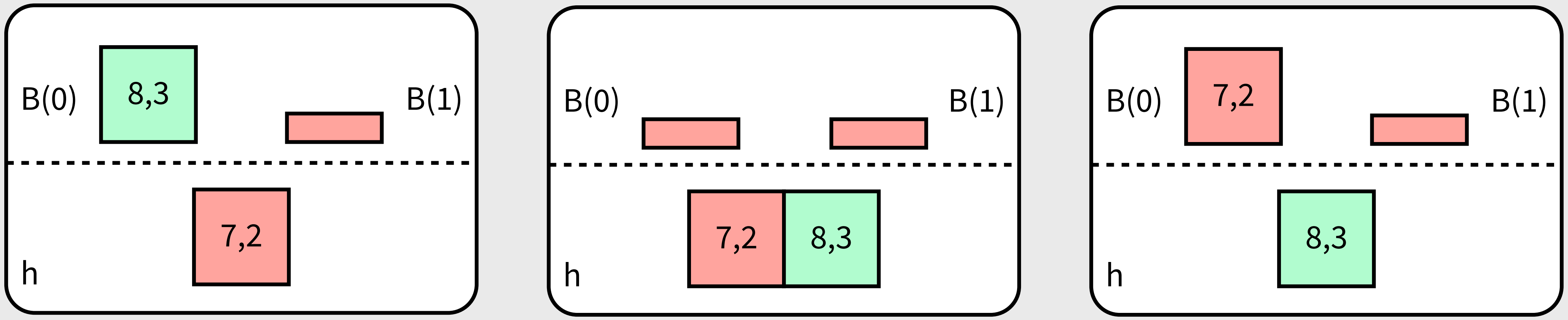
$7 \rightsquigarrow_1 2$



Buffered writes

$7 \rightsquigarrow_0 2 * 8 \rightsquigarrow_0 3$





Buffered writes

$$7 \rightsquigarrow_0 2 * \underline{7} \rightsquigarrow_0 3$$

Inconsistent!

Spatial separating conjunction can't:

describe writes to the same location;

describe writes in any particular order.

Temporal separation

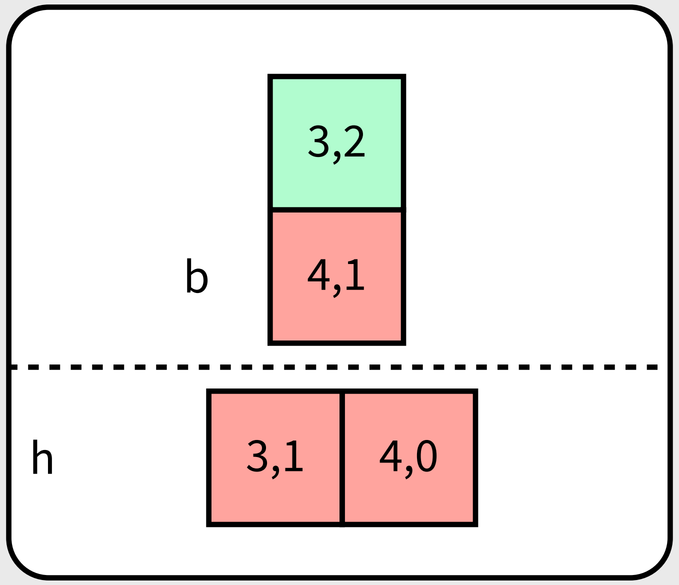
$P \lhd Q$: **temporal separating conjunction** of P and Q.

Describes *ordered* sequences of writes to *non-disjoint* addresses.

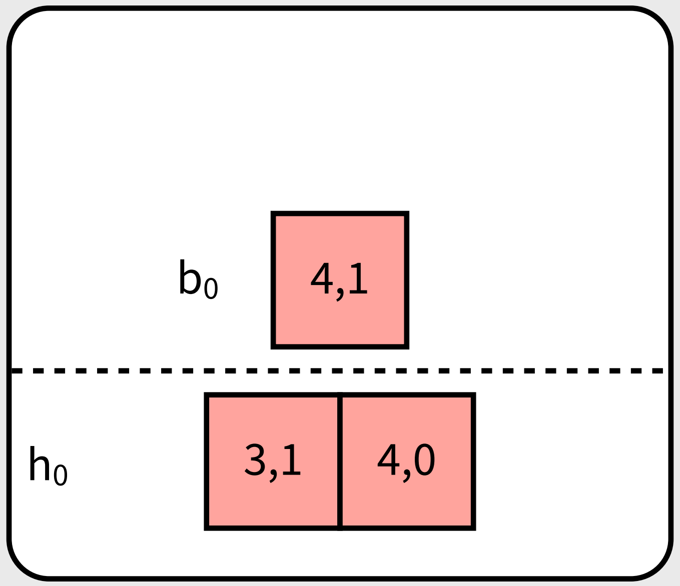
Separates x86 states according to time:

writes described by P must *occur before* writes described by Q.

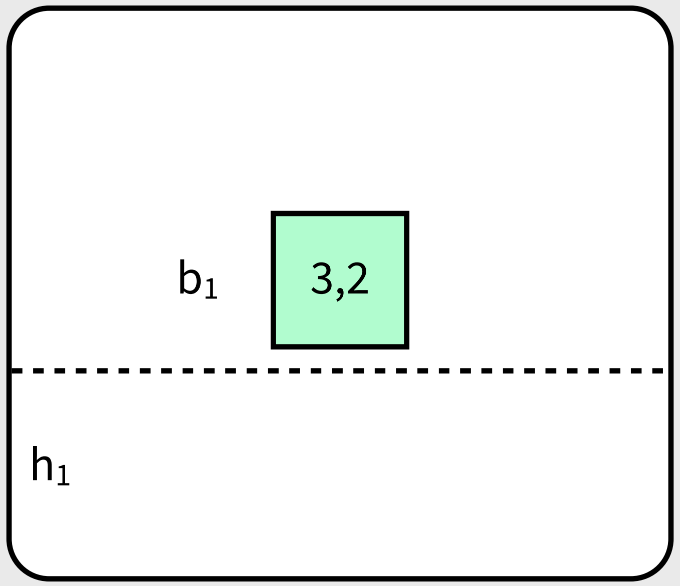
Temporal separation



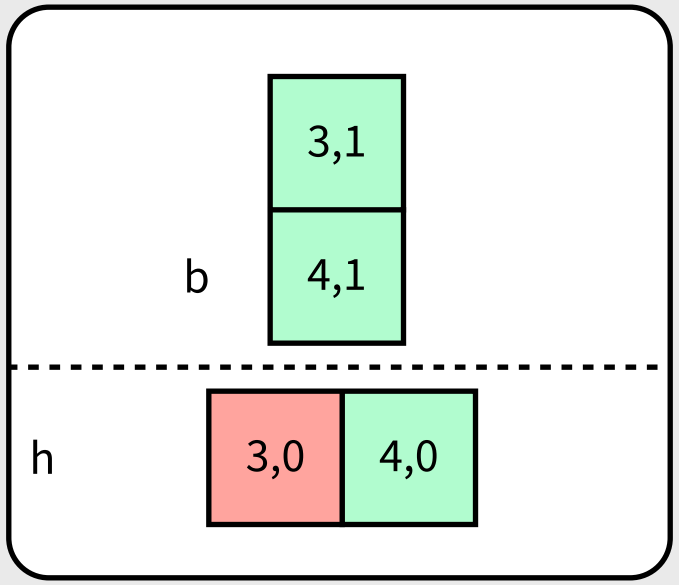
(h,b)



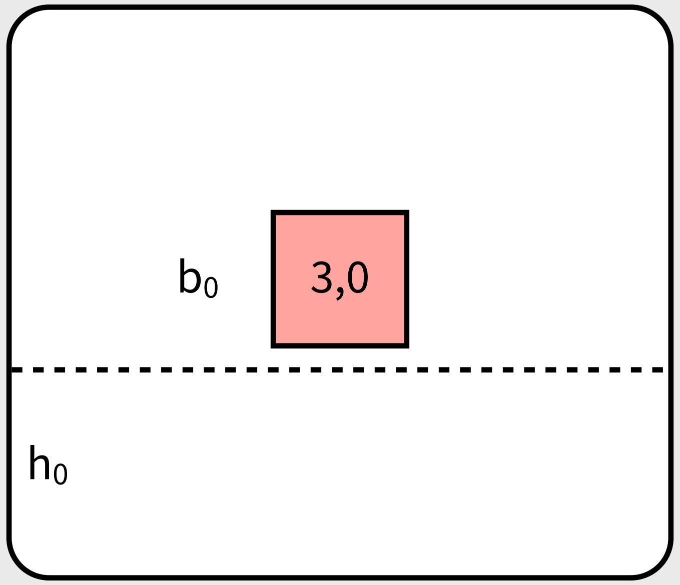
$(h_0,b_0) \lhd (h_1,b_1)$



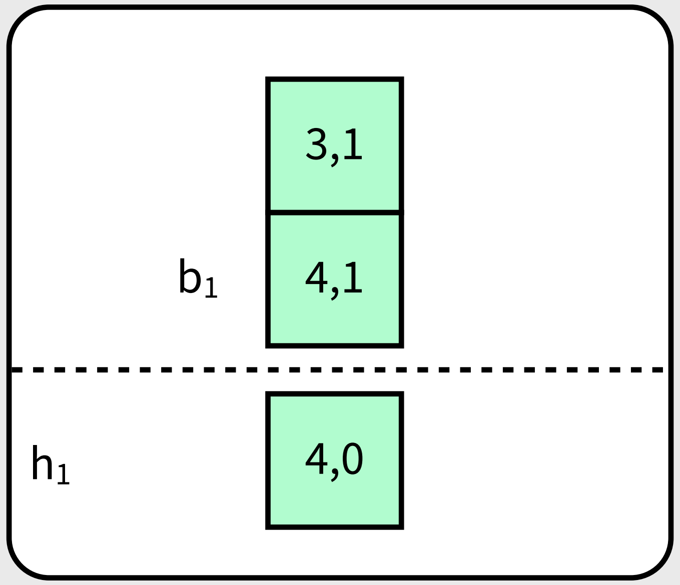
Temporal separation



(h,b)

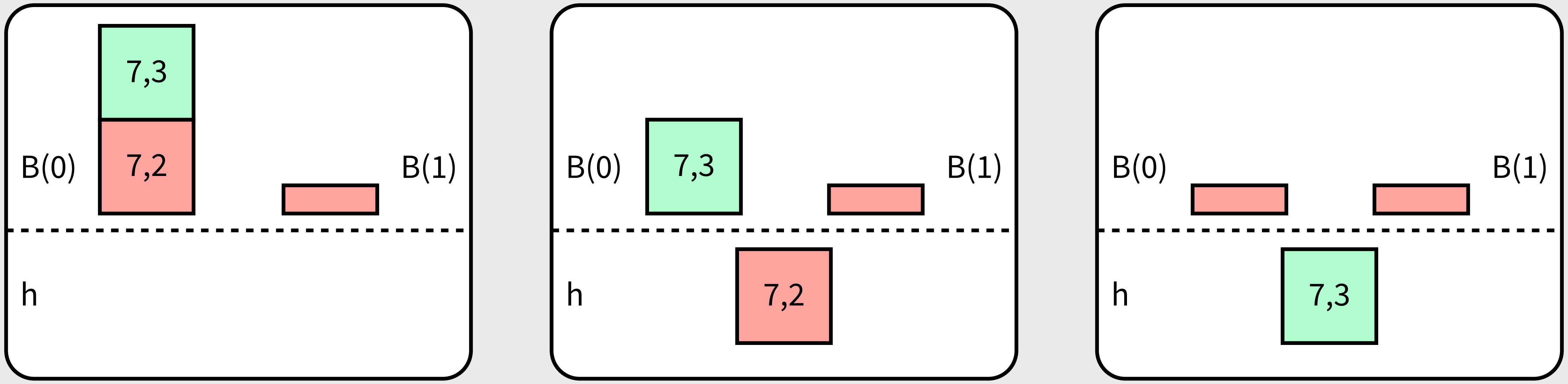


$(h_0,b_0) \quad \lhd \quad (h_1,b_1)$



Write sequences

$$7 \rightsquigarrow_0 2 \lhd 7 \rightsquigarrow_0 3$$



Temporal locality

Commands are local in space **and time**.

Consider a load $x \coloneqq [7]$:

Assigns to x value of the *most recent* write to address 7.

Earlier writes are irrelevant.

Temporal frame rule:

$$\frac{J \vdash \{\, P \,\}\; c\; \{\, Q \,\}}{J \vdash \{\, R \triangleleft P \,\}\; c\; \{\, R \triangleleft Q \,\}}$$

Strong temporal separation

$P \blacktriangleleft Q$: strong temporal separating conjunction.

Separates in both time and space;

$P \blacktriangleleft Q \triangleq (P * Q) \wedge (P \triangleleft Q)$

Strong temporal frame rule:

$$\frac{J \vdash \{P\}\, c\, \{Q\}}{J \vdash \{R \blacktriangleleft P\}\, c\, \{R \blacktriangleleft Q\}}$$

Load and store

Load axiom:

Good: $J \vdash \{ e \rightsquigarrow_i f \}\ x \coloneqq [e]_i\ \{ e \rightsquigarrow_i f \wedge x = e \}$

Better: $J \vdash \{ e \rightsquigarrow_i f \blacktriangleleft P \}\ x \coloneqq [e]_i\ \{ (e \rightsquigarrow_i f \blacktriangleleft P) \wedge x = e \}$

Store axiom:

Good: $J \vdash \{ e \rightsquigarrow_i e' \}\ [e] \coloneqq f_i\ \{ e \rightsquigarrow_i e' \triangleleft e \rightsquigarrow_i f \}$

Better: $J \vdash \{ e \rightsquigarrow_i e' \triangleleft P \}\ [e] \coloneqq f_i\ \{ e \rightsquigarrow_i e' \triangleleft P \triangleleft e \rightsquigarrow_i f \}$

Conclusion

Contributions:

- A programming language with an x86-like model.
- An assertion logic with an x86-like model.
- A CSL-style logic for local reasoning about x86-like programs.
  - (Examples indicate reasoning might not be significantly harder than in CSL.)

Lots of work left!

- Some important meta-theory remains (e.g., soundness).
- Proof theory of specifications *must* be strengthened.

Thank you

Advisors:

Warren Hunt and J Moore.

Committee, etc.:

Josh Berdine, Allen Emerson, Don Fussell, Tony Hoare and Mohamed Gouda.

Everyone else!

Barrier assertions

emp : empty state

$P * \mathbf{emp} \equiv P \lhd \mathbf{emp} \equiv P \equiv \mathbf{emp} \lhd P \equiv \mathbf{emp} * P$

$\mathbf{bar}_i$: result of flushing ith write buffer

$P \lhd \mathbf{bar}_i$: like P but with empty ith buffer

Expresses fence axiom:

$J \vdash \{\,\mathbf{emp}\,\}\ \mathrm{fence}_i\ \{\,\mathbf{bar}_i\,\}$

$J \vdash \{\,P\,\}\ \mathrm{fence}_i\ \{\,P \lhd \mathbf{bar}_i\,\}$

Lock assertions

lock$_i$ describes states in which processor i holds lock.

$i \neq j \wedge$ (**lock**$_i$ $*$ **lock**$_j$) : inconsistent because lock is exclusive.

$i \neq j \wedge$ (**lock**$_i$ $* \; e \rightsquigarrow_j f$) : buffered write *only* because j is blocked by i.

Lock axioms

Good:

$J \vdash \{\,\mathbf{emp}\,\}\ \mathsf{lock}_i\ \{\,\mathbf{lock}_i\}$

$J \vdash \{\,\mathbf{lock}_i\,\}\ \mathsf{unlock}_i\ \{\,\mathbf{emp}\,\}$

Better:

$J \vdash \{\,\mathbf{emp}\,\}\ \mathsf{lock}_i\ \{\,\mathbf{lock}_i * \mathbf{bar}_i\,\}$

$J \vdash \{\,\mathbf{lock}_i\,\}\ \mathsf{unlock}_i\ \{\,\mathbf{bar}_i\,\}$

Concurrency

Accessing shared state:

$$\frac{\mathbf{emp} \vdash \{\, J * P * \mathbf{lock}_i \,\}\; c \;\{\, J * Q * \mathbf{lock}_i \,\}}{J \vdash \{\, P * \mathbf{lock}_i \,\}\; c \;\{\, Q * \mathbf{lock}_i \,\}}$$

Concurrent composition:

$$\frac{J \vdash \{\, P \,\}\; c \;\{\, Q \,\} \quad \text{and} \quad J \vdash \{\, P' \,\}\; c' \;\{\, Q' \,\}}{J \vdash \{\, P * P' \,\}\; c \| c' \;\{\, Q * Q' \,\}}$$

Sharing private state:

$$\frac{J \vdash \{\, P \,\}\; c \;\{\, Q \,\}}{\mathbf{emp} \vdash \{\, J * P \,\}\; c \;\{\, J * Q \,\}}$$

Closure

Assertions denote sets that are closed under flushing:

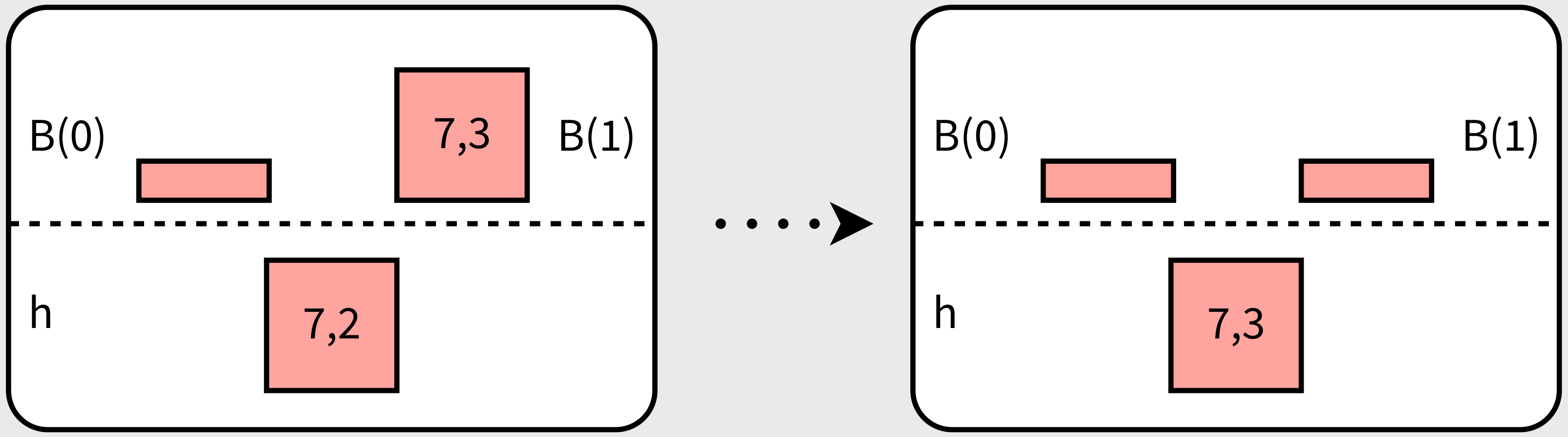
if $\sigma \vDash P$ and σ can flush writes to yield σ' then also $\sigma' \vDash P$.

Nondeterministic flushing is hidden by the logic;
no explicit reasoning about flushing.

Important for soundness:

$$J \vdash \{ P \}\ \mathsf{skip}_i\ \{ P \}$$

Races and disjunction



$x \coloneqq [7]_0$ is a racy load; a true post-condition is: $x = 2 \lor x = 3$

Can we use the disjunction rule to reason about racy loads?

$$\frac{J \vdash \{\, P \,\}\; c\; \{\, Q \,\} \quad \text{and} \quad J \vdash \{\, P' \,\}\; c\; \{\, Q \,\}}{J \vdash \{\, P \lor P' \,\}\; c\; \{\, Q \,\}}$$

No: the former state, alone, is not closed under flushing!