Modeling systems via register machines

for the verification of weak memory models

Elli Anastasiadi [1,2] Samuel Grahn [1]

[1]Department of Information Technology, Uppsala University, Sweden
firstname.lastname@it.uu.se

[2]Department of Computer Science, Aalborg University, Denmark
firstname.lastname@mail.dk

November 2024

UPPSALA
UNIVERSITET

Weak Memory Models

Why WMMs?

- Memory access is slow, so hardware designers have implemented *caches*.
- Distributed systems that pass information about the system using messages.

Weak Memory Models

Why WMMs?

- Memory access is slow, so hardware designers have implemented *caches*.
- Distributed systems that pass information about the system using messages.

Writes are not immediately visible to all possible readers (threads, systems, et.c.) Any such memory model is called *weak*. Notable examples include TSO, RA, ARM.

Weak Memory Models

Why WMMs?

- Memory access is slow, so hardware designers have implemented *caches*.
- Distributed systems that pass information about the system using messages.

Writes are not immediately visible to all possible readers (threads, systems, et.c.) Any such memory model is called *weak*. Notable examples include TSO, RA, ARM.

Does a given implementation satisfy a given WMM?

Weak Memory Models

Why WMMs?

- Memory access is slow, so hardware designers have implemented *caches*.
- Distributed systems that pass information about the system using messages.

Writes are not immediately visible to all possible readers (threads, systems, et.c.) Any such memory model is called *weak*. Notable examples include TSO, RA, ARM.

Does a given implementation satisfy a given WMM?

Undecidable in general[1]

Weak Memory Models

Why WMMs?

- Memory access is slow, so hardware designers have implemented *caches*.
- Distributed systems that pass information about the system using messages.

Writes are not immediately visible to all possible readers (threads, systems, et.c.) Any such memory model is called *weak*. Notable examples include TSO, RA, ARM.

Does a given implementation satisfy a given WMM?

Undecidable in general[1]

Simplify the model!

Register Machines

Assume a set Θ of threads, a set $\mathcal{V}$ of variables, and a set $\mathtt{Regs}$ of registers, the values of which range over some domain $\mathcal{D}$.

Register Machines

Assume a set Θ of threads, a set $\mathcal{V}$ of variables, and a set $\mathtt{Regs}$ of registers, the values of which range over some domain $\mathcal{D}$.

Definition (Operation)

- $(\mathsf{W}, \theta, x, a)$ – Thread θ writes to variable x, storing the value in register a.
- $(\mathsf{R}, \theta, x, a)$ – Thread θ reads from the variable x, and gets the value stored in register a.
- $a := b$ – The value of register b is copied into register a.

Register Machines

Assume a set Θ of threads, a set $\mathcal{V}$ of variables, and a set $\mathtt{Regs}$ of registers, the values of which range over some domain $\mathcal{D}$.

Definition (Operation)

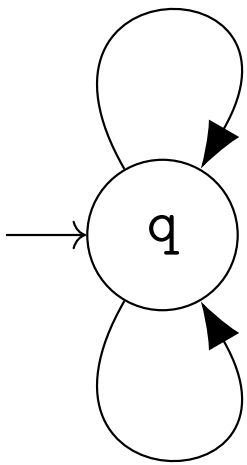
- $(\mathsf{W}, \theta, x, a)$ – Thread θ writes to variable x, storing the value in register a.
- $(\mathsf{R}, \theta, x, a)$ – Thread θ reads from the variable x, and gets the value stored in register a.
- $a := b$ – The value of register b is copied into register a.

Definition (Register Machine)

A *register machine* $\mathcal{M}$ is a tuple $\langle Q, q_{\mathtt{init}}, \Delta \rangle$, where Q is the (finite) set of states, $q_{\mathtt{init}} \in Q$ is the initial state, and Δ is the finite set of transitions, where each $t \in \Delta$ is of the form $\langle q, \mathsf{o}, q' \rangle$ where $q, q' \in Q$ are states and o is an operation.

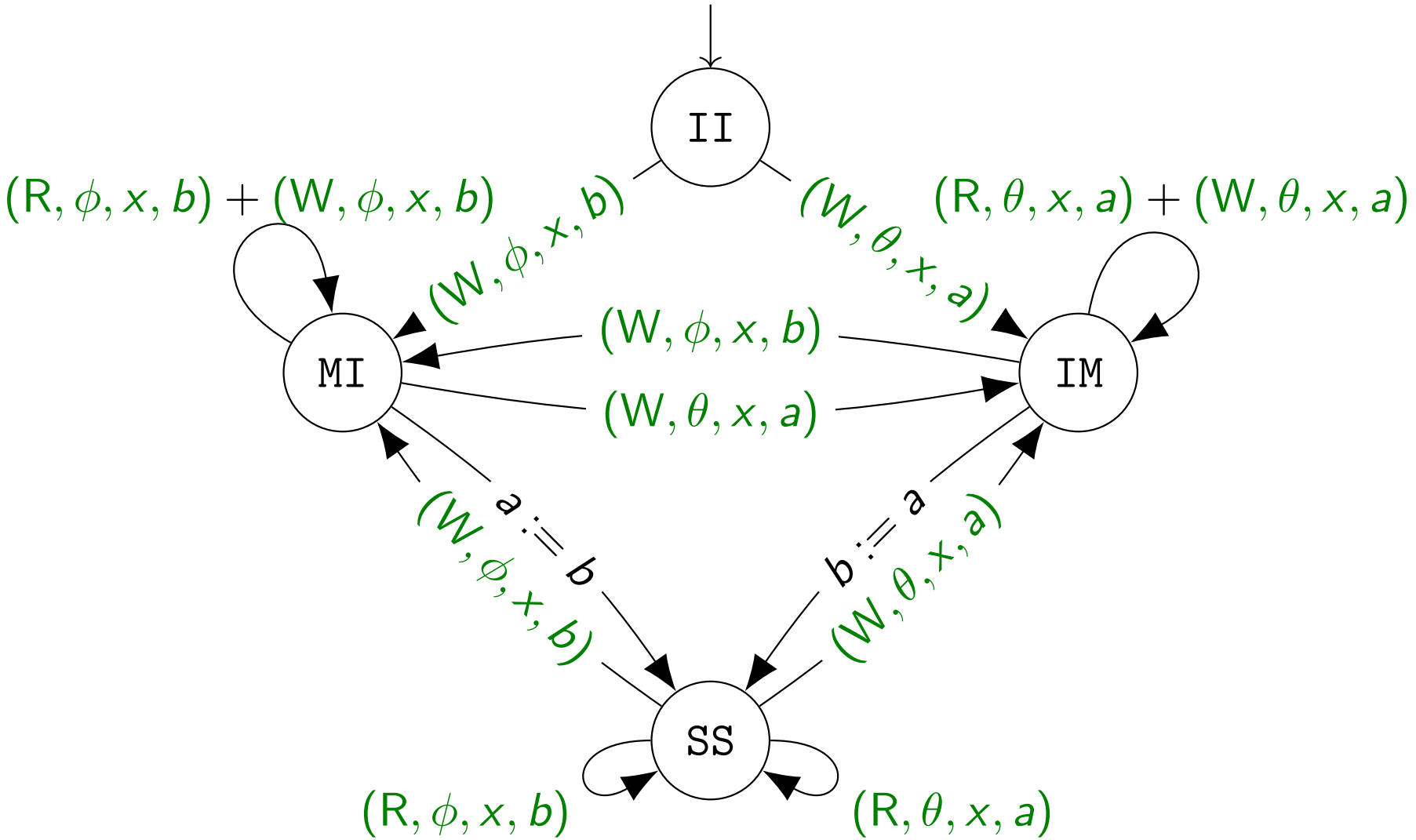
Example: Instantaneous visibility

$(\mathrm{R}, \theta, x, a) + (\mathrm{W}, \theta, x, a)$



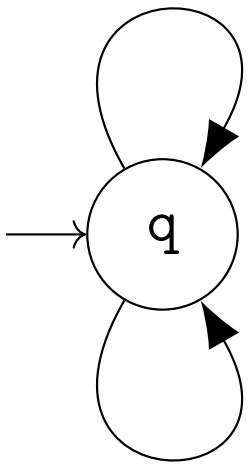
$(\mathrm{R}, \phi, x, a) + (\mathrm{W}, \phi, x, a)$

Example: MSI Protocol



Thread-local Memory

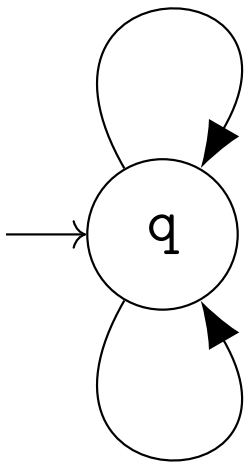
$$(\mathrm{R}, \theta, x, a) + (\mathrm{W}, \theta, x, a)$$



$$(\mathrm{R}, \phi, x, a) + (\mathrm{W}, \phi, x, a)$$

Thread-local Memory

$$(\mathrm{R}, \theta, x, a) + (\mathrm{W}, \theta, x, a)$$

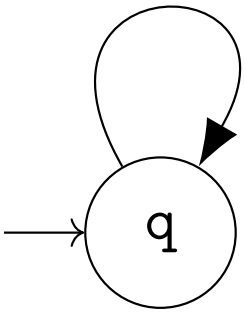


$$(\mathrm{R}, \phi, x, a) + (\mathrm{W}, \phi, x, a)$$

Writes are instantly visible to all threads!

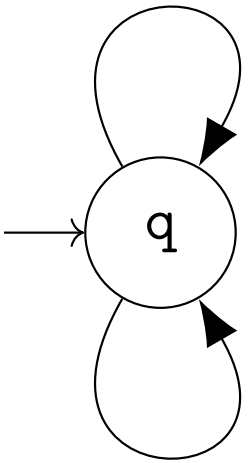
Thread-local Memory

$$(\mathsf{R} \vee \mathsf{W}, \theta, x, a_\theta) + (\mathsf{R} \vee \mathsf{W}, \phi, x, a_\phi)$$



Thread-local Memory

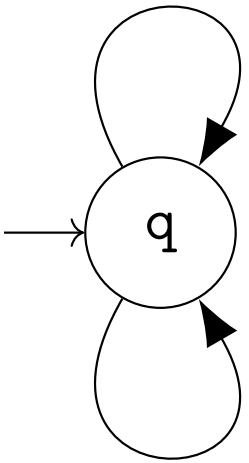
$$(\mathsf{R} \vee \mathsf{W}, \theta, x, a_\theta) + (\mathsf{R} \vee \mathsf{W}, \phi, x, a_\phi)$$



$$a_\theta := a_\phi + a_\phi := a_\theta$$

Thread-local Memory

$$(\mathsf{R} \vee \mathsf{W}, \theta, x, a_\theta) + (\mathsf{R} \vee \mathsf{W}, \phi, x, a_\phi)$$



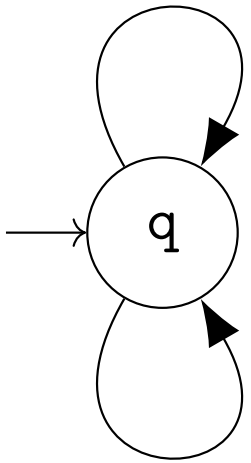
$$a_\theta := a_\phi + a_\phi := a_\theta$$

Overwritten writes may return!

$$\begin{aligned}(\mathsf{W}, \theta, x, a_\theta) &\to a_\phi := a_\theta \to (\mathsf{R}, \phi, x, a_\phi) \\ &\to (\mathsf{W}, \phi, x, a_\phi) \to a_\phi := a_\theta \\ &\to (\mathsf{R}, \phi, x, a_\phi)\end{aligned}$$

Thread-local Memory

$$(\mathrm{R} \vee \mathrm{W}, \theta, x, a_\theta) + (\mathrm{R} \vee \mathrm{W}, \phi, x, a_\phi)$$



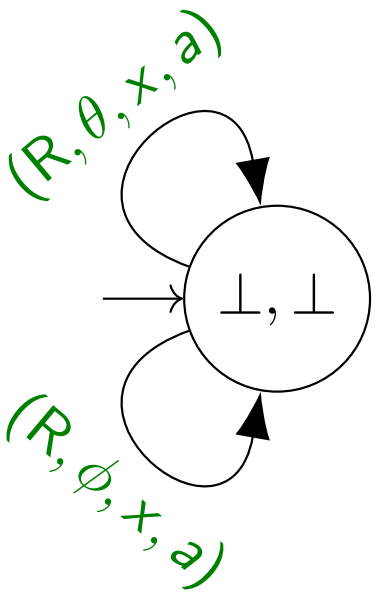
$$a_\theta := a_\phi + a_\phi := a_\theta$$

Overwritten writes may return!

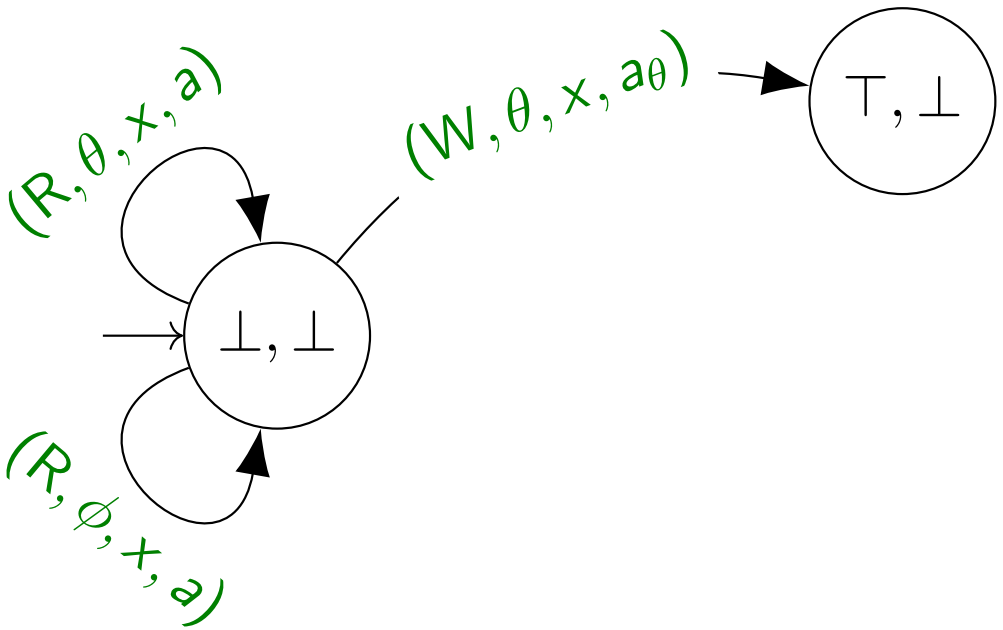
$$\begin{aligned}(\mathrm{W}, \theta, x, a_\theta) &\to a_\phi := a_\theta \to (\mathrm{R}, \phi, x, a_\phi) \\ &\to (\mathrm{W}, \phi, x, a_\phi) \to a_\phi := a_\theta \\ &\to (\mathrm{R}, \phi, x, a_\phi)\end{aligned}$$

Solution: Encode information about whether a written value has been passed to shared memory.

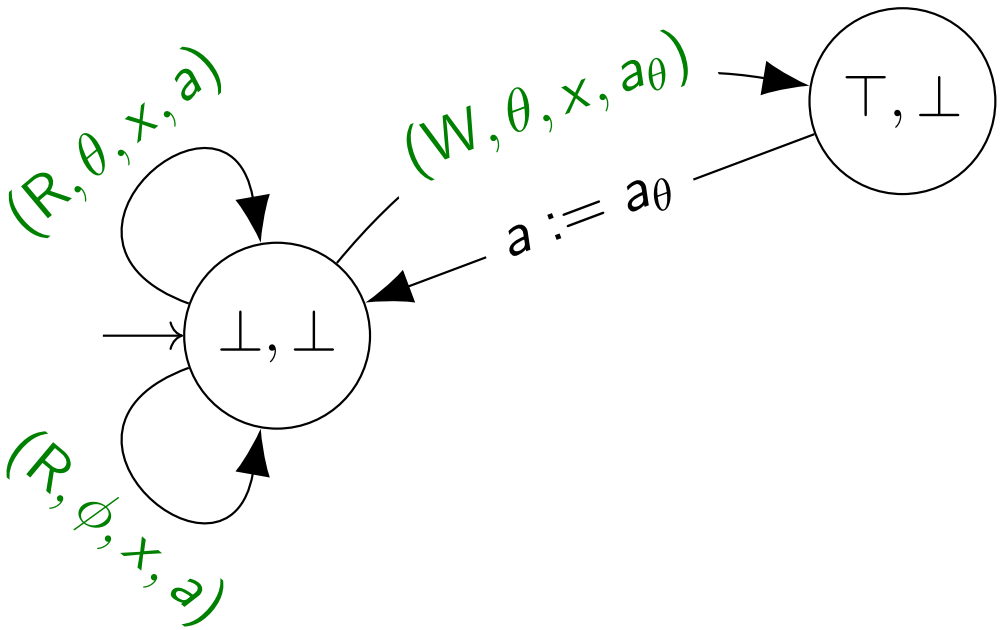
Thread-local and Shared Memory



Thread-local and Shared Memory

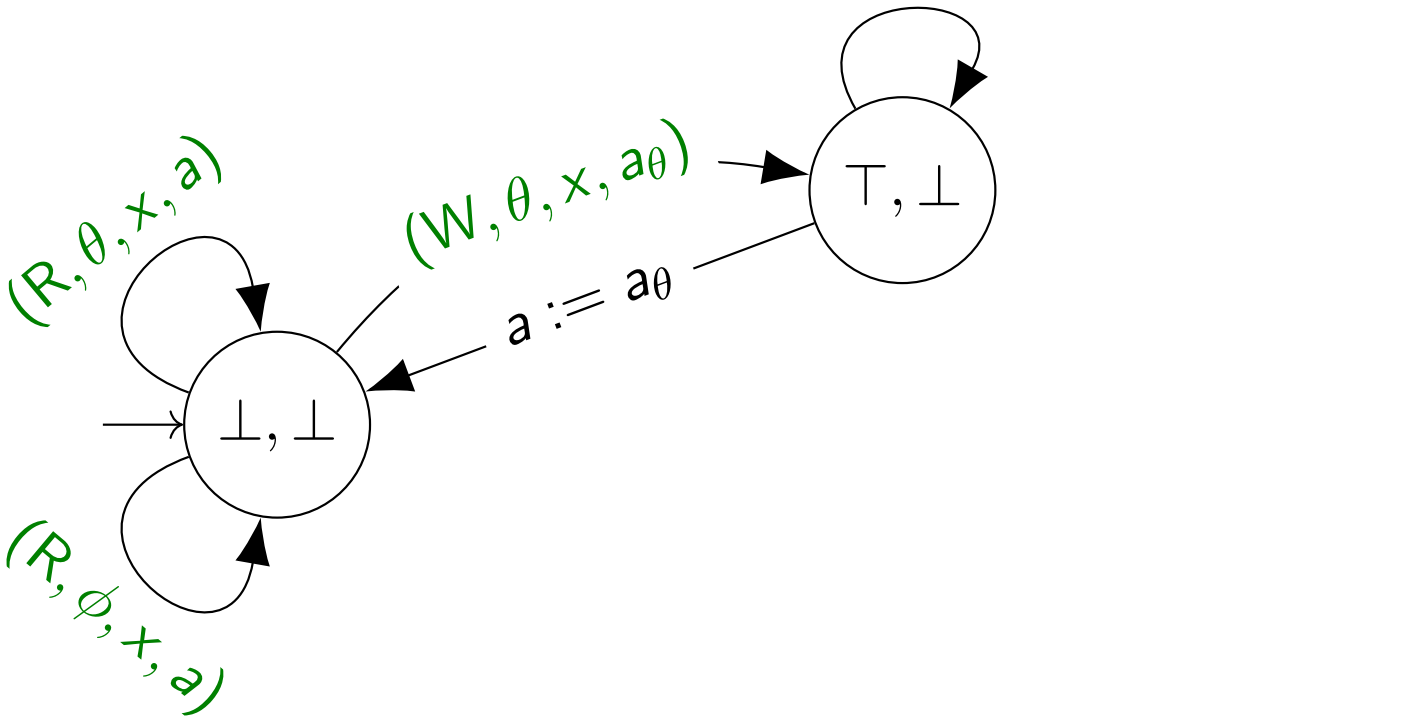


Thread-local and Shared Memory

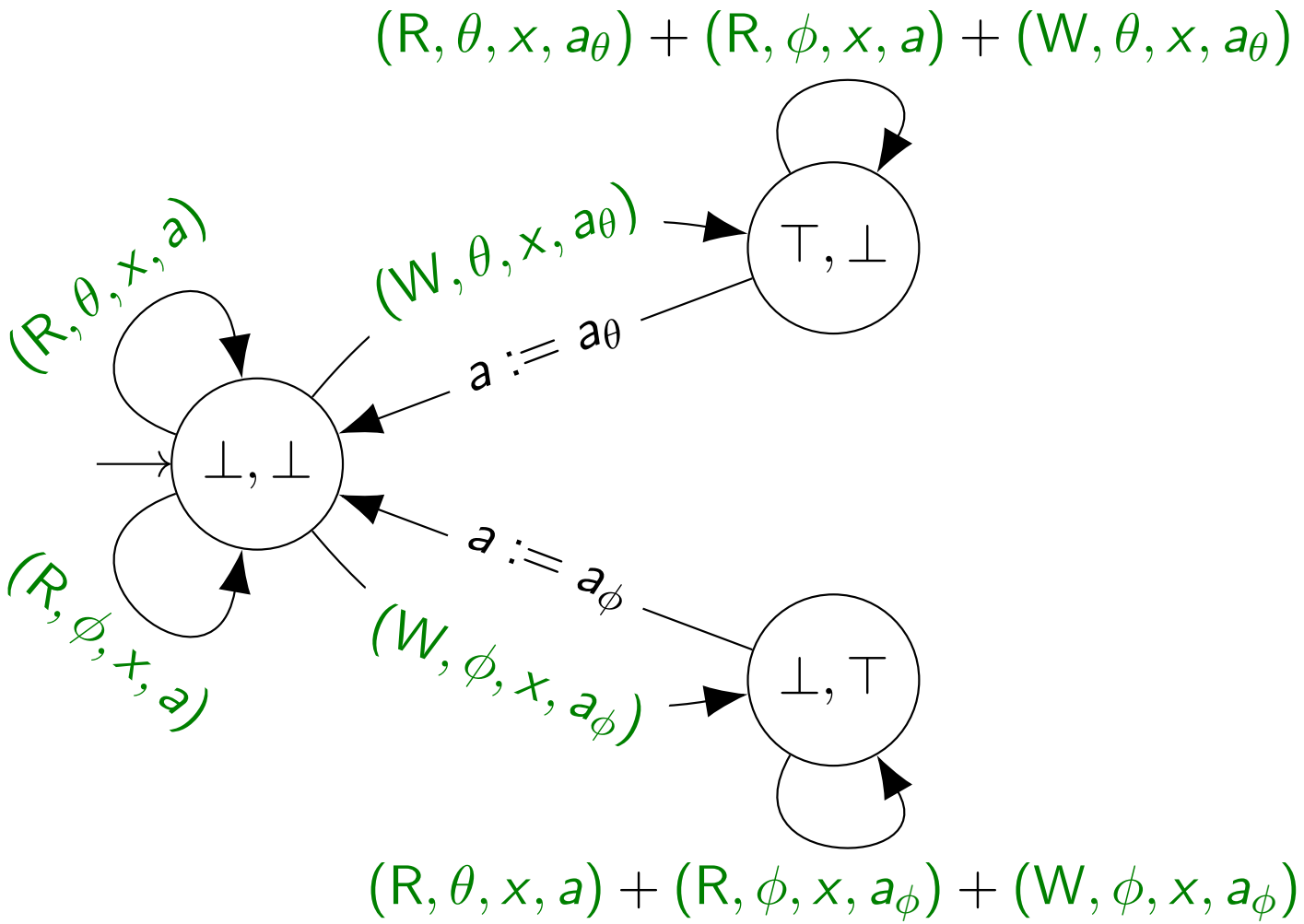


Thread-local and Shared Memory

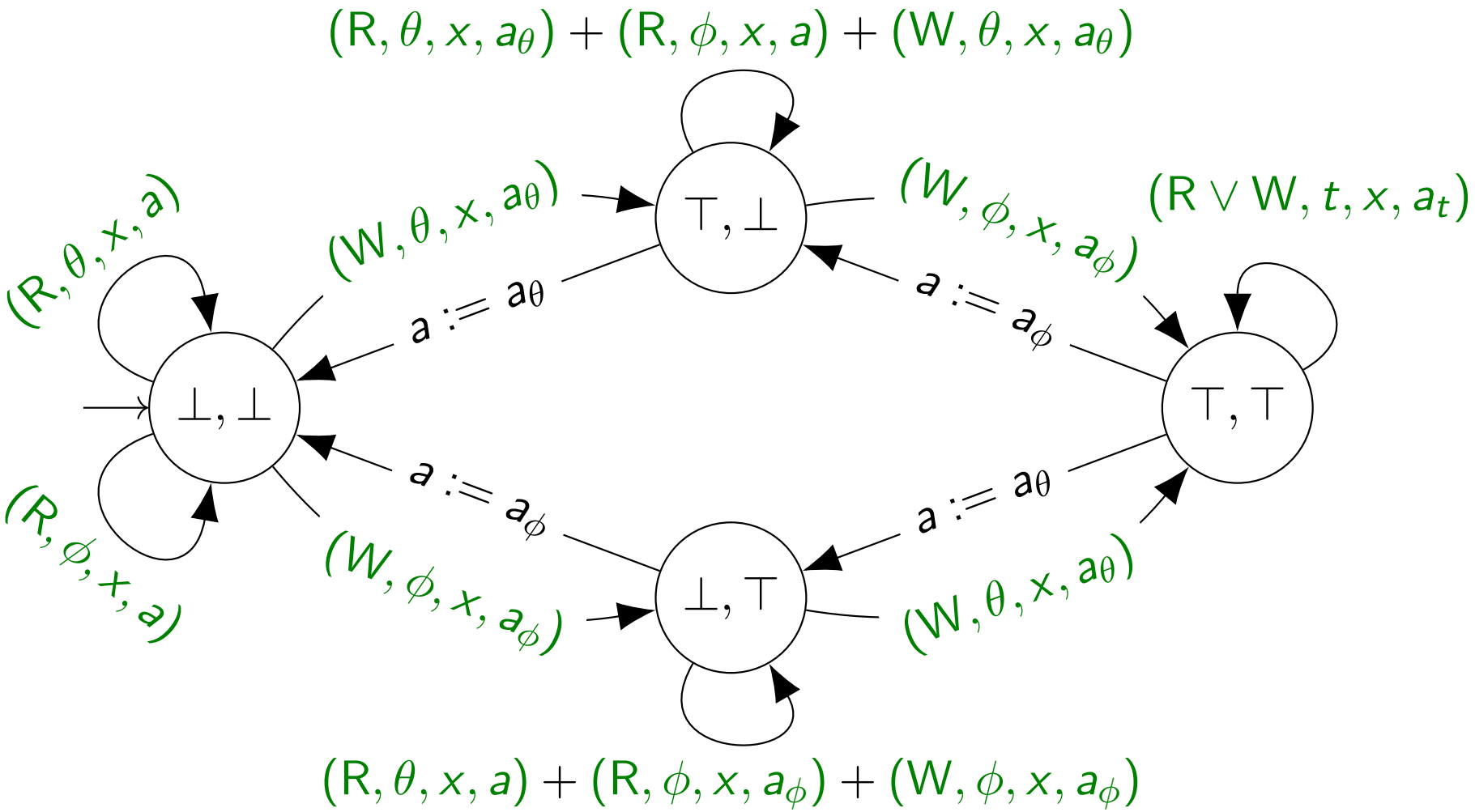
$$(\mathsf{R}, \theta, x, a_\theta) + (\mathsf{R}, \phi, x, a) + (\mathsf{W}, \theta, x, a_\theta)$$



Thread-local and Shared Memory



Thread-local and Shared Memory



Buffers

When the effect of some action of a system is delayed for some participants, we can (sometimes) model it using buffers.

Buffers

When the effect of some action of a system is delayed for some participants, we can (sometimes) model it using buffers.

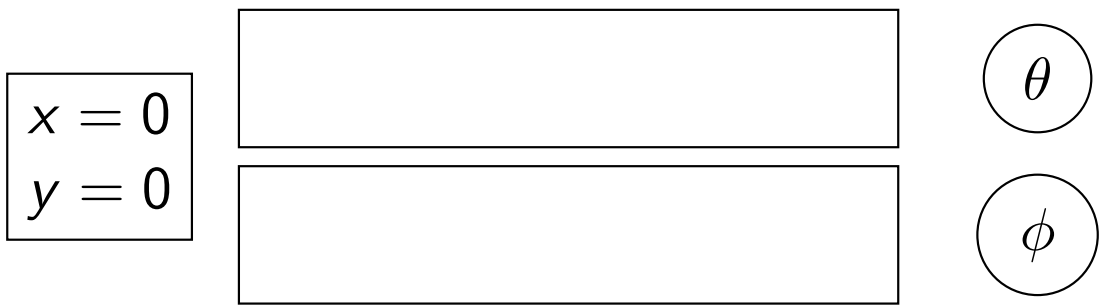
- Writes that are not immediately visible to all threads (e.g. `TSO` write- or load buffer semantics)

Buffers

When the effect of some action of a system is delayed for some participants, we can (sometimes) model it using buffers.

- Writes that are not immediately visible to all threads (e.g. `TSO` write- or load buffer semantics)
- Delays due to traveling time in distributed systems (e.g. message queues)

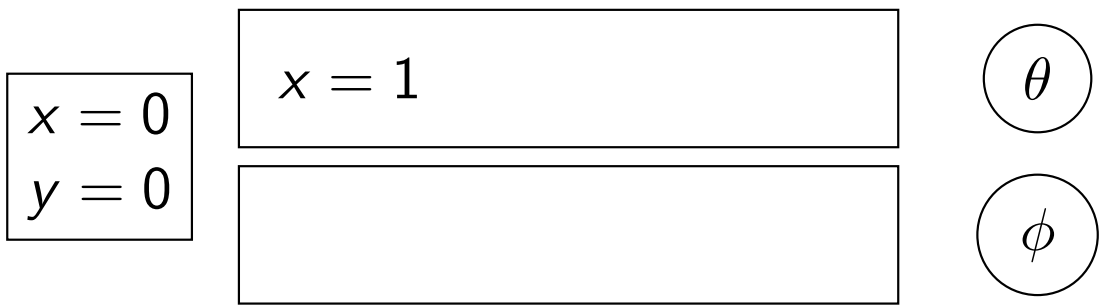
TSO-style Store Buffers



Write: Append to own buffer

Read: Rightmost occurrence in own buffer, otherwise memory

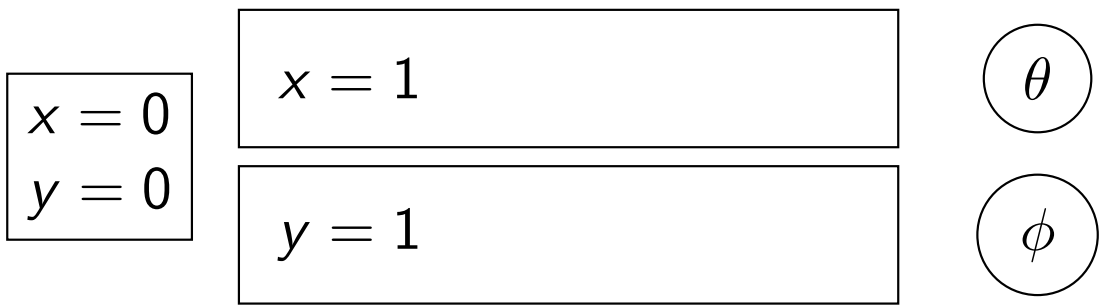
TSO-style Store Buffers



Write: Append to own buffer

Read: Rightmost occurrence in own buffer, otherwise memory

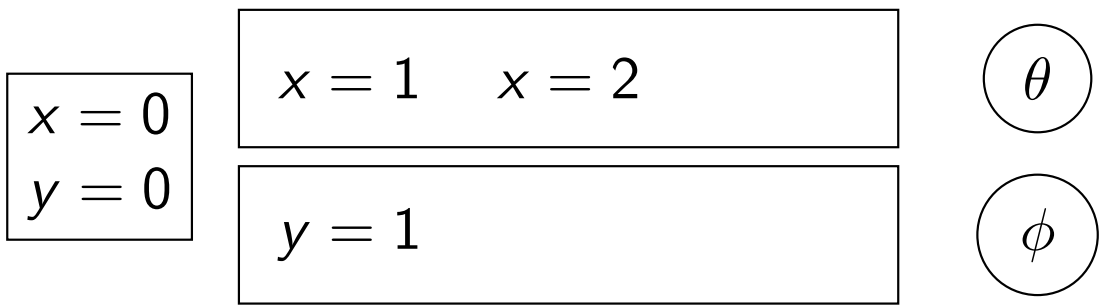
TSO-style Store Buffers



Write: Append to own buffer

Read: Rightmost occurrence in own buffer, otherwise memory

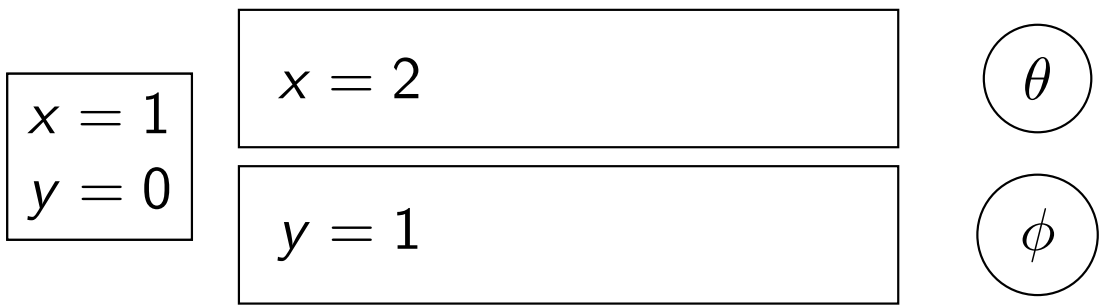
TSO-style Store Buffers



Write: Append to own buffer

Read: Rightmost occurrence in own buffer, otherwise memory

TSO-style Store Buffers



Write: Append to own buffer

Read: Rightmost occurrence in own buffer, otherwise memory

TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

- Variables: $x, y, z, \ldots$

TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

- Variables: $x, y, z, \ldots$
- Buffers: $B^{\theta}, B^{\phi}, \ldots$

TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

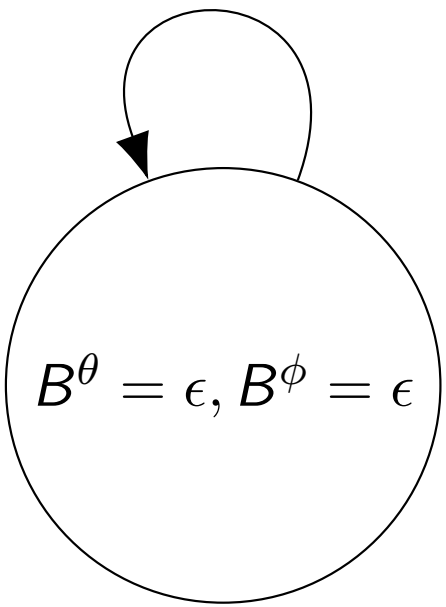
- Variables: $x, y, z, \ldots$
- Buffers: $B^{\theta}, B^{\phi}, \ldots$
- Registers: $x_{mem}, y_{mem}, \ldots, B^{\theta}_1, \ldots, B^{\theta}_n, B^{\phi}_1, \ldots$

TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

- Variables: $x, y, z, \ldots$
- Buffers: $B^\theta, B^\phi, \ldots$
- Registers: $x_{mem}, y_{mem}, \ldots, B^\theta_1, \ldots, B^\theta_n, B^\phi_1, \ldots$

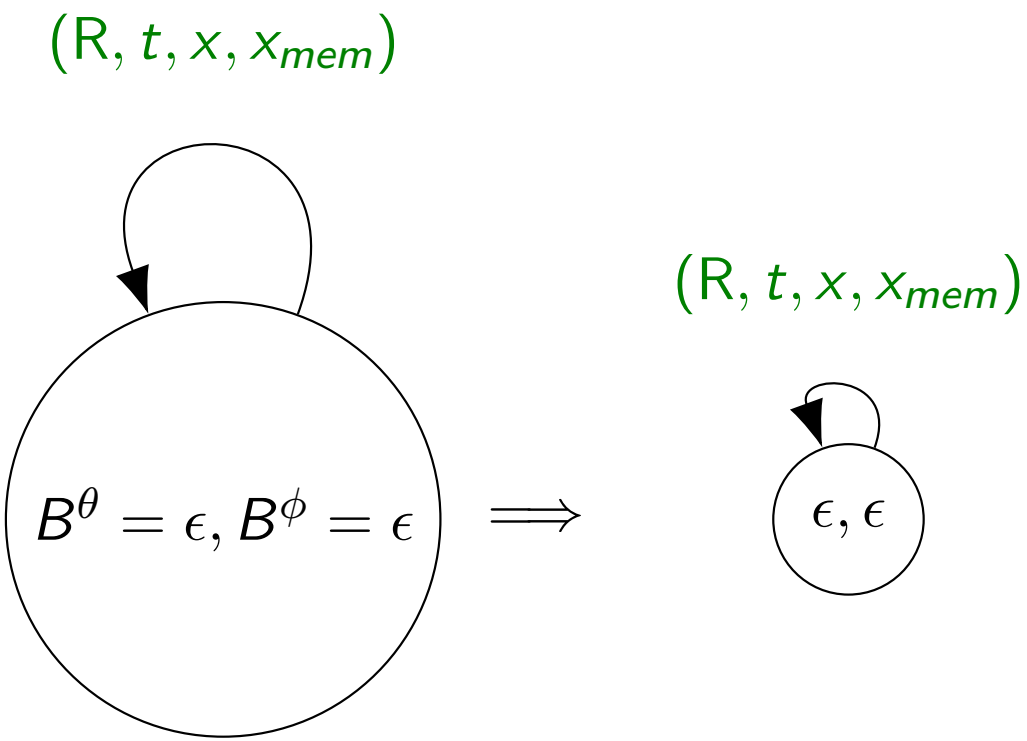
$(\mathrm{R}, t, x, x_{mem})$



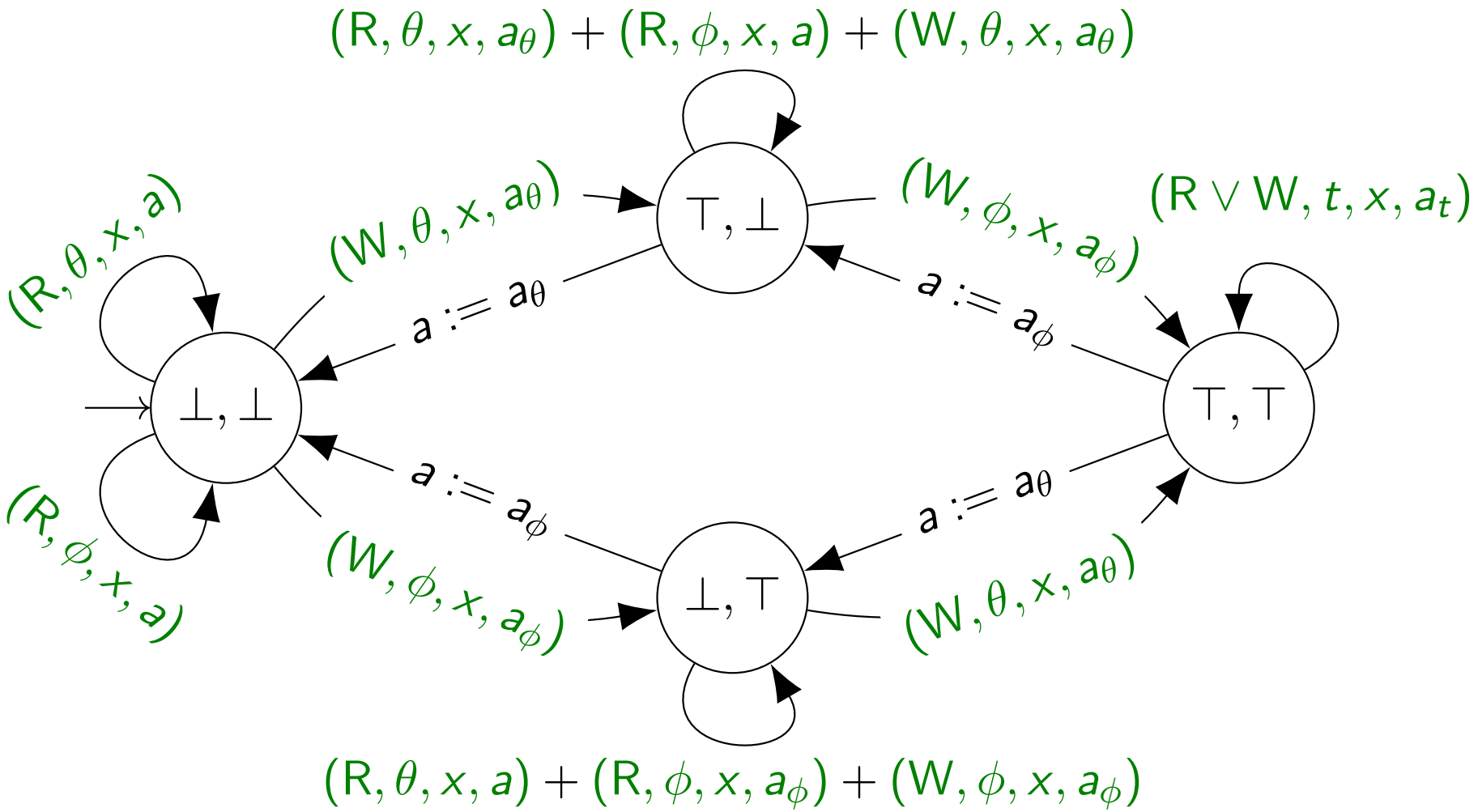
TSO-style Store Buffers

Encoding TSO-style store buffers buffers as register machines

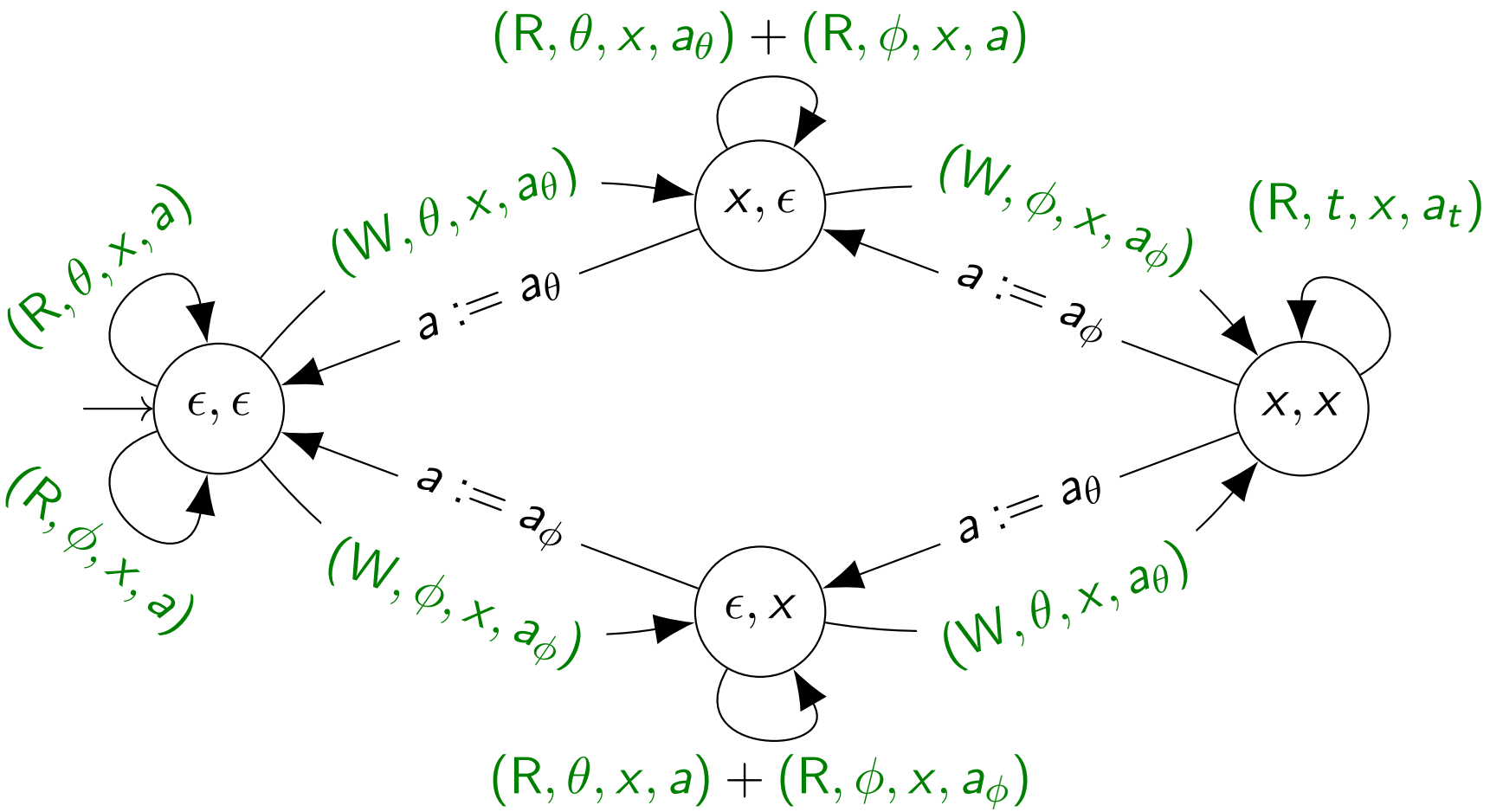
- Variables: $x, y, z, \ldots$
- Buffers: $B^\theta, B^\phi, \ldots$
- Registers: $x_{mem}, y_{mem}, \ldots, B^\theta_1, \ldots, B^\theta_n, B^\phi_1, \ldots$



TSO-style Store Buffers

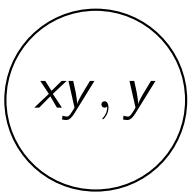


TSO-style Store Buffers



TSO-style Store Buffers: Read/Write

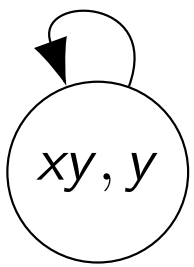
Writing and reading



TSO-style Store Buffers: Read/Write

Writing and reading

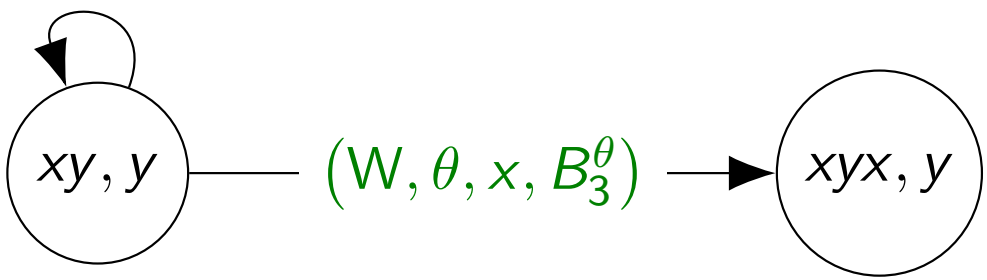
$$\begin{aligned}&\left(\mathrm{R}, \theta, x, B_1^\theta\right)+\\&\left(\mathrm{R}, \theta, y, B_2^\theta\right)+\\&\left(\mathrm{R}, \phi, y, B_1^\phi\right)+\\&\left(\mathrm{R}, \phi, x, x_{mem}\right)\end{aligned}$$



TSO-style Store Buffers: Read/Write

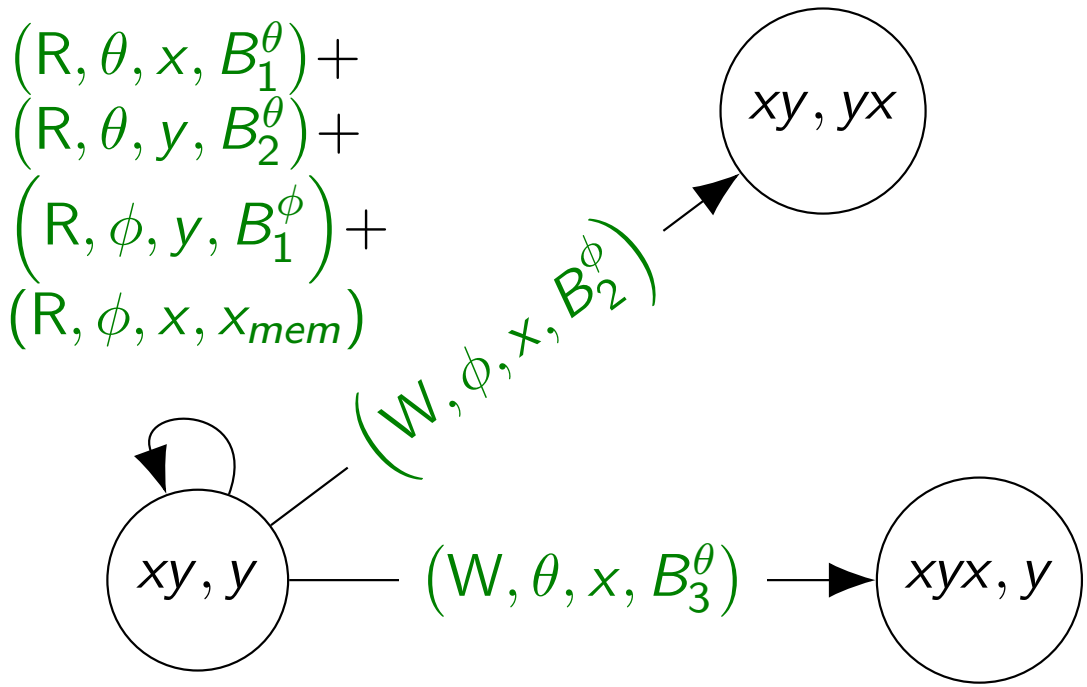
Writing and reading

$$
\begin{aligned}
&\left(\mathrm{R}, \theta, x, B_1^{\theta}\right) + \\
&\left(\mathrm{R}, \theta, y, B_2^{\theta}\right) + \\
&\left(\mathrm{R}, \phi, y, B_1^{\phi}\right) + \\
&\left(\mathrm{R}, \phi, x, x_{mem}\right)
\end{aligned}
$$

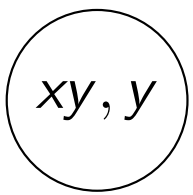


TSO-style Store Buffers: Read/Write

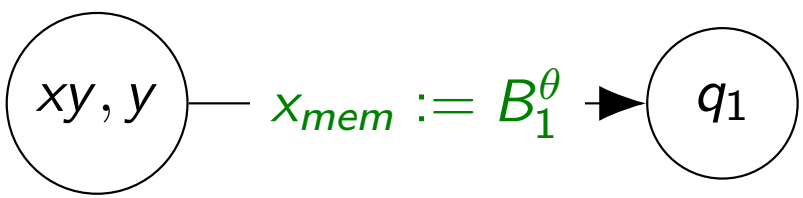
Writing and reading



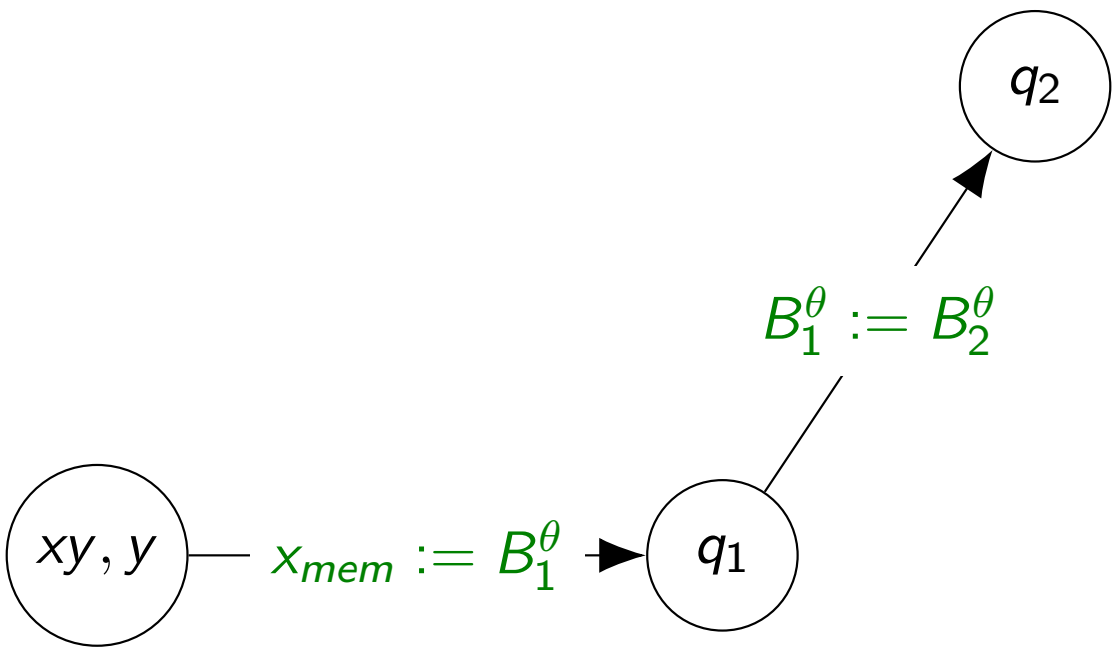
TSO-style Store Buffers: Handling message



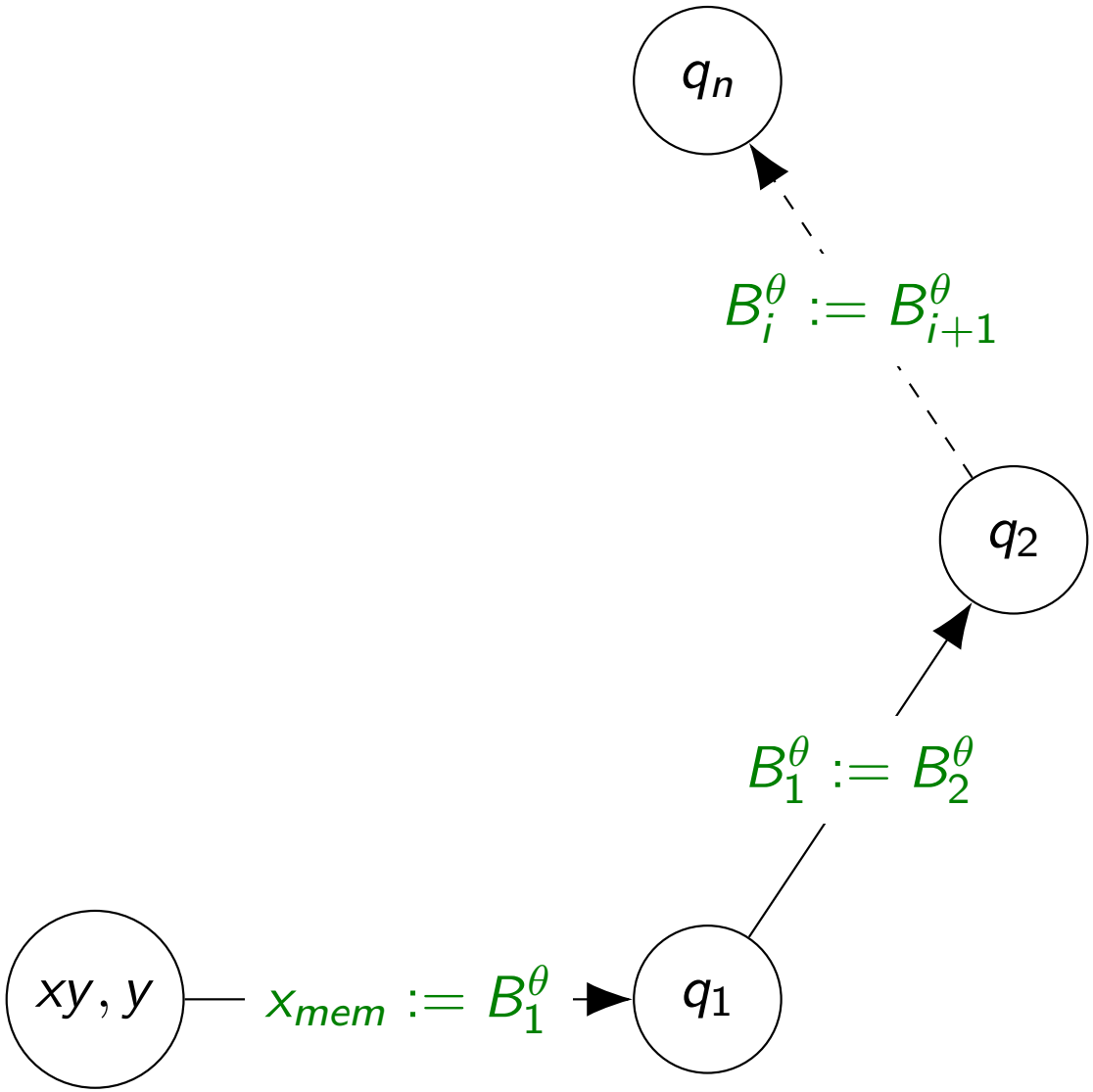
TSO-style Store Buffers: Handling message



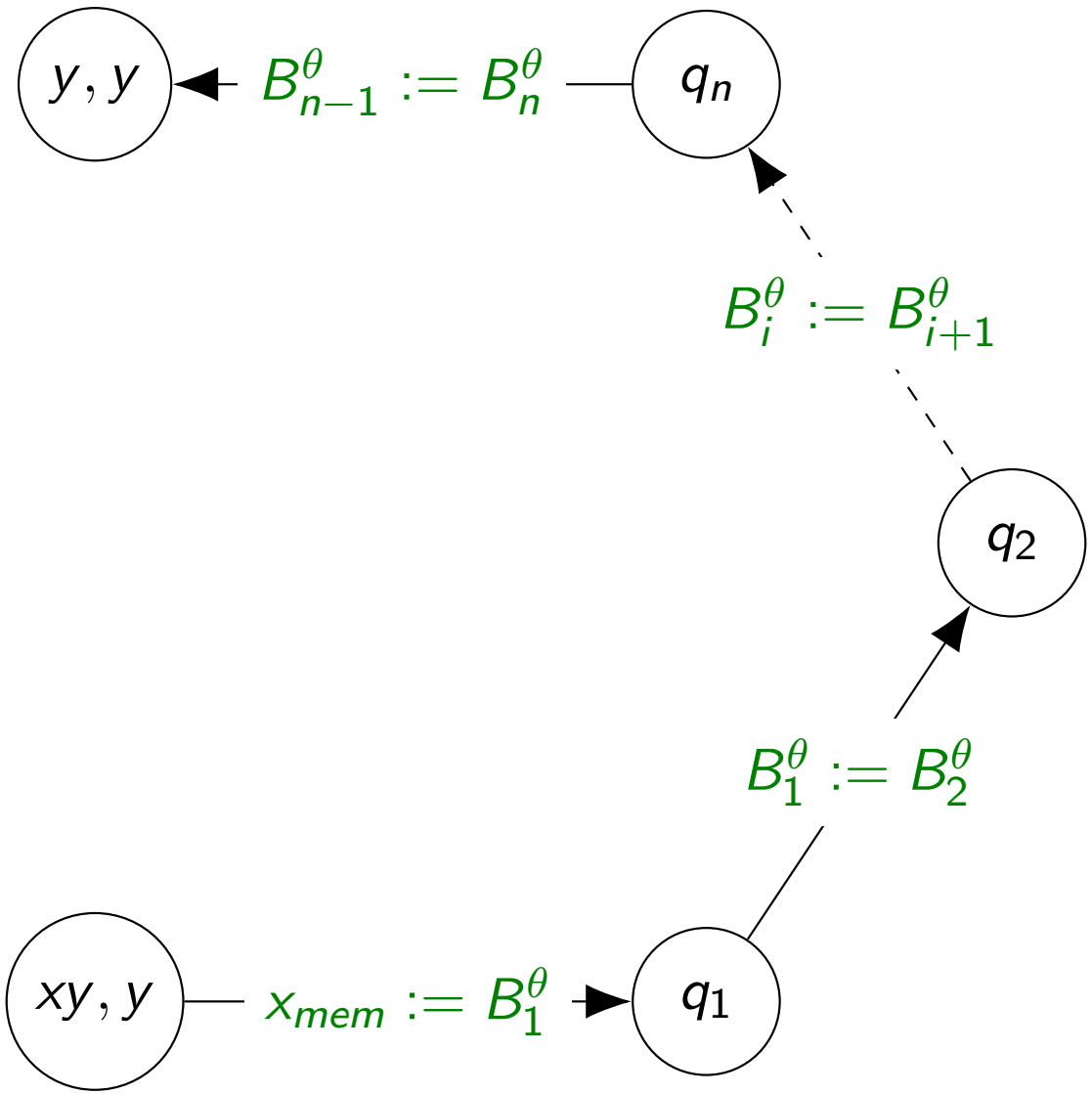
TSO-style Store Buffers: Handling message



TSO-style Store Buffers: Handling message



TSO-style Store Buffers: Handling message



Fences

A fence is an instruction in which each thread waits for the buffers to be empty before doing anything. Assume a fence from a state q to a state q'.

Fences

A fence is an instruction in which each thread waits for the buffers to be empty before doing anything. Assume a fence from a state q to a state q'.

- If the buffers are empty in q, we *only* have the "nondeterministic copies" available from q.

Fences

A fence is an instruction in which each thread waits for the buffers to be empty before doing anything. Assume a fence from a state q to a state q'.

- If the buffers are empty in q, we *only* have the "nondeterministic copies" available from q.
- Otherwise, we have a dummy transition $q \xrightarrow{a:=a} q'$.

Conclusion

We model buffers as part of the state. Two weaknesses:

Conclusion

We model buffers as part of the state. Two weaknesses:

1. Requires bounded buffer sizes and thread counts – usually the case in real systems!

Conclusion

We model buffers as part of the state. Two weaknesses:

1. Requires bounded buffer sizes and thread counts – usually the case in real systems!
2. Exponential growth (state explosion) – not ideal, but OK.

Conclusion

We model buffers as part of the state. Two weaknesses:

1. Requires bounded buffer sizes and thread counts – usually the case in real systems!
2. Exponential growth (state explosion) – not ideal, but OK.

However: We have decidability for more memory models, and we can still model useful systems!

References

[1] Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza.
On verifying causal consistency.
In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 626–638. ACM, 2017.