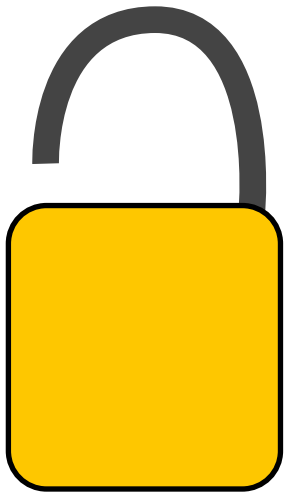
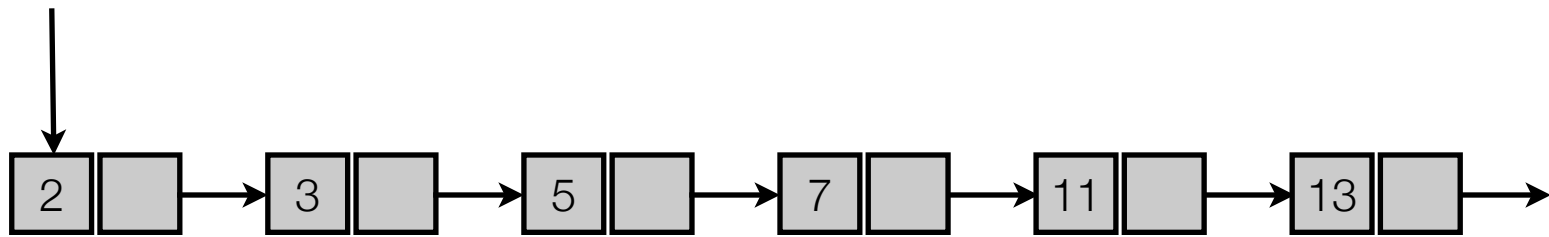


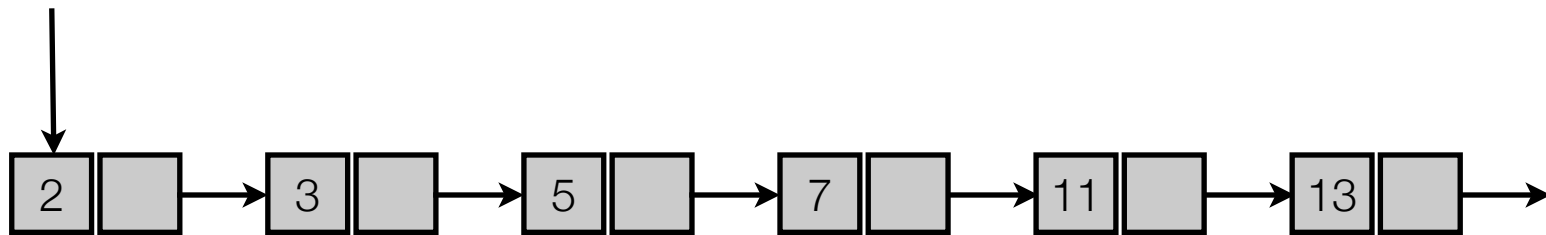
RGSep

Viktor Vafeiadis

Coarse-grain locking



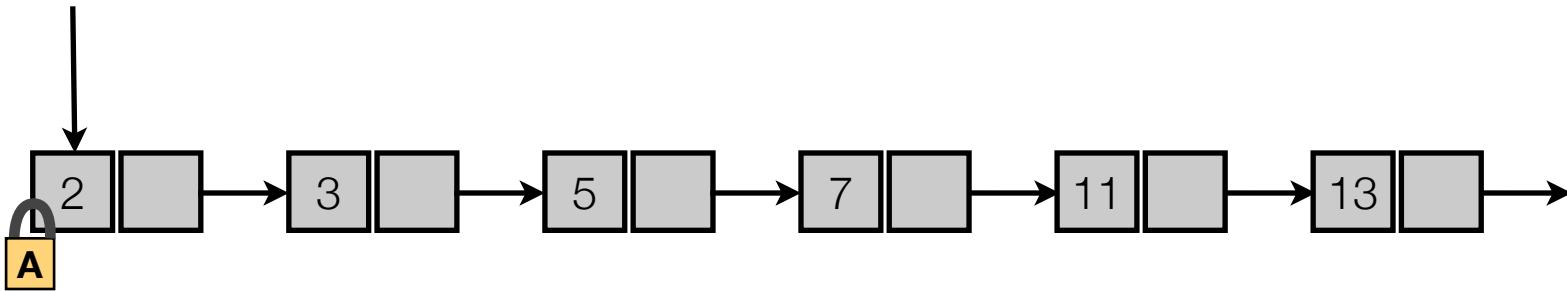
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

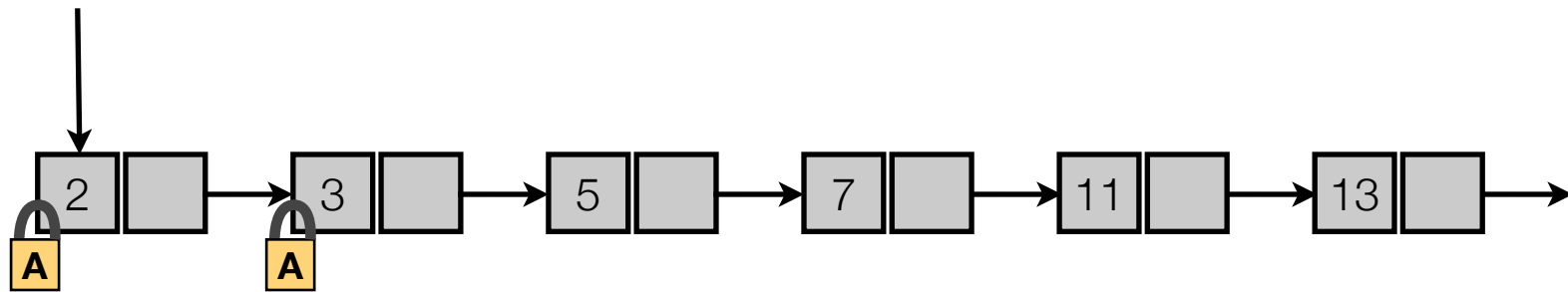
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

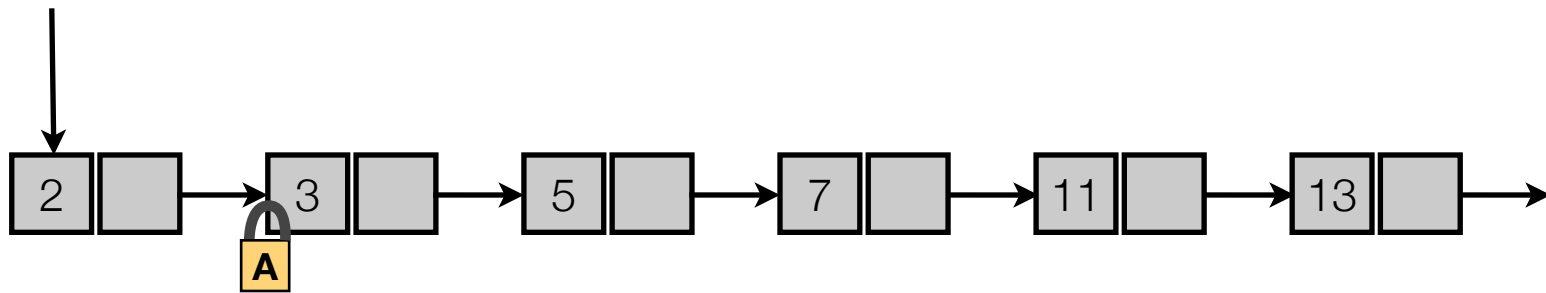
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

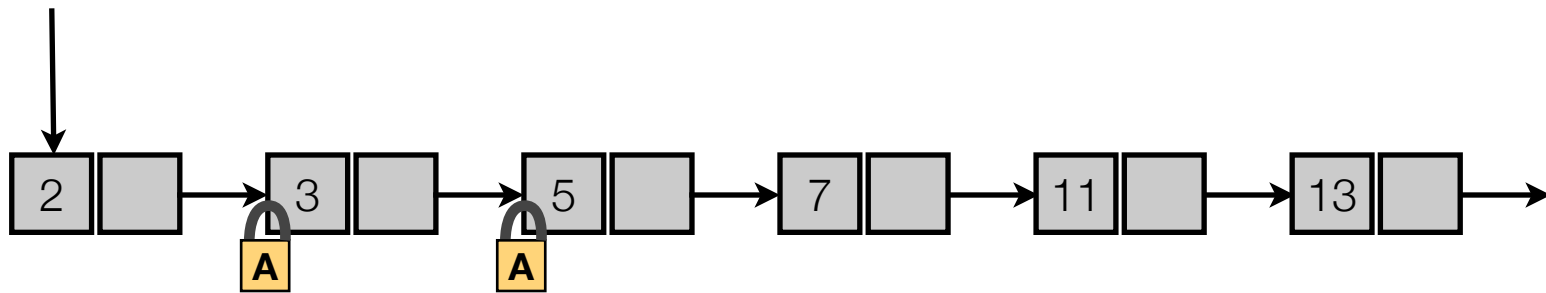
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

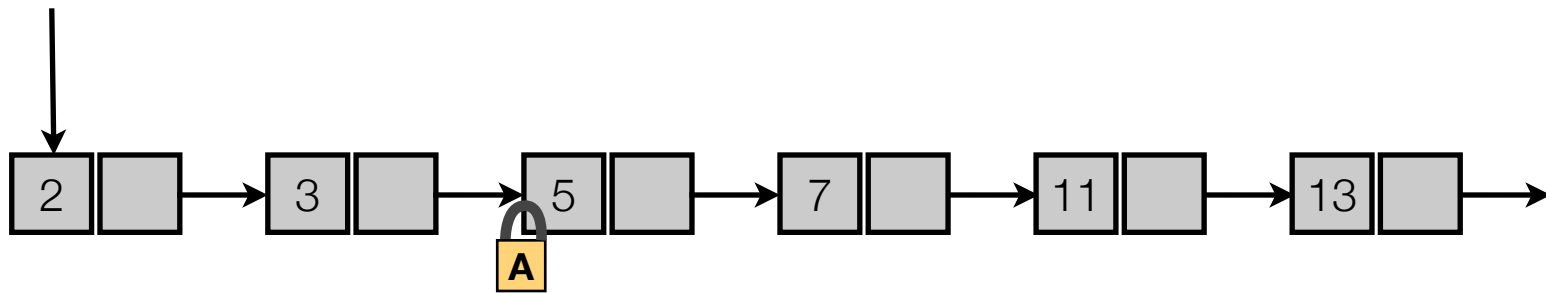
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

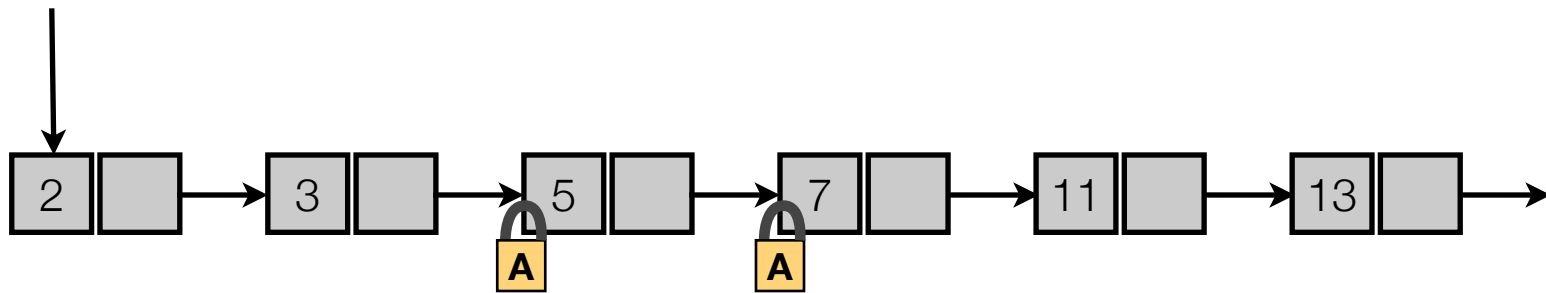
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

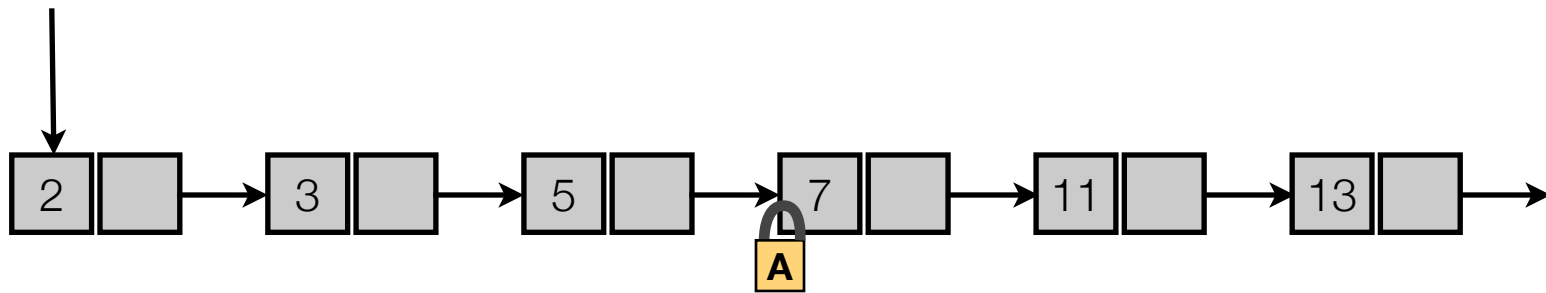
Fine-grain locking (pessimistic)



One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

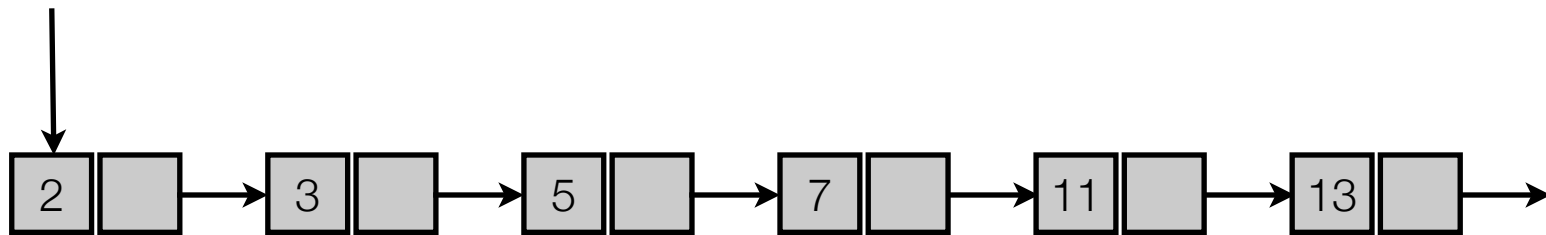
Fine-grain locking (pessimistic)



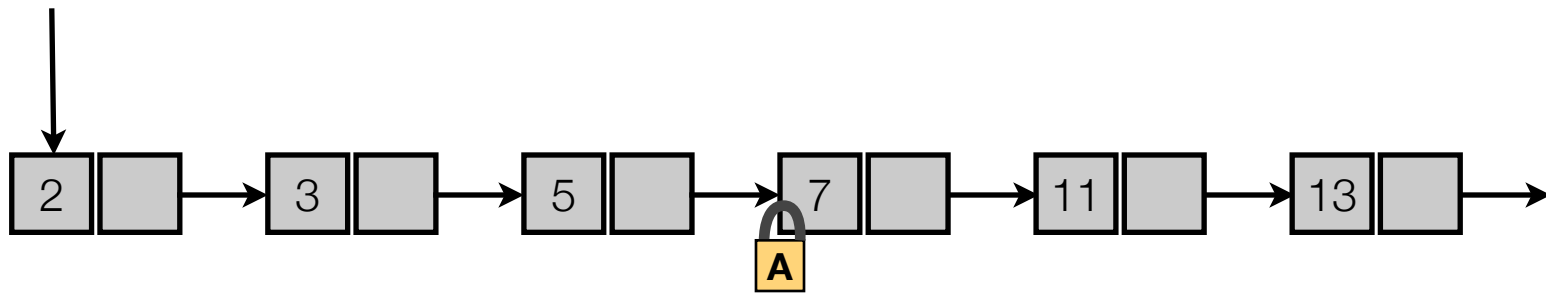
One lock per node:

- Traversals acquire locks in a “hand over hand” fashion.
- If node is locked, we can add a node after it.
- If two adjacent nodes are locked, we can delete the second.

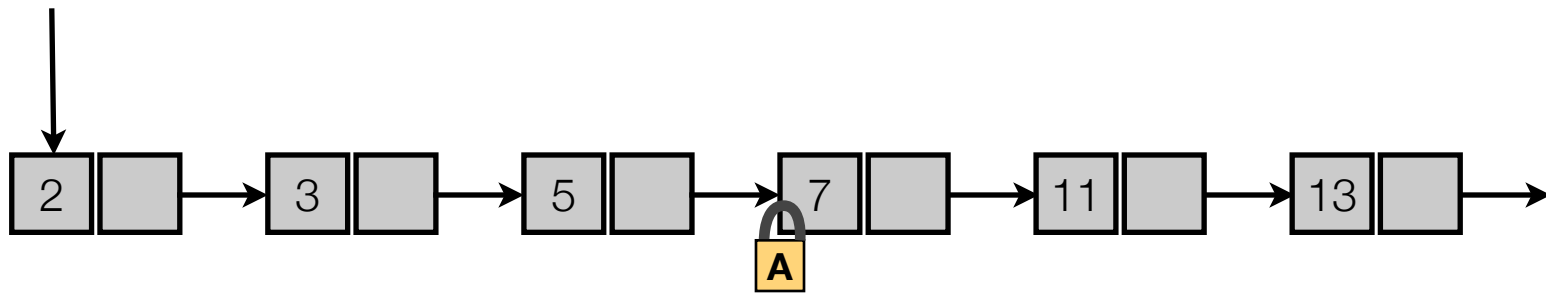
Fine-grain locking (optimistic)



Fine-grain locking (optimistic)



Fine-grain locking (optimistic)

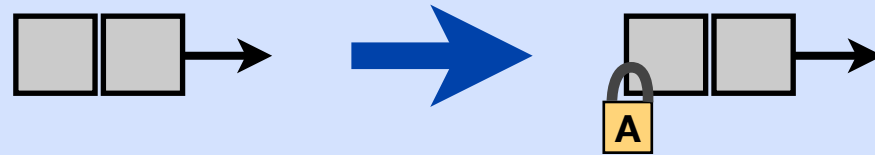


Re-traverse the list OR perform deletions in two steps

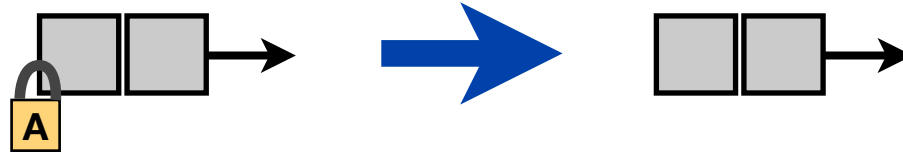
Leaks memory: cannot dispose deleted nodes.

Actions (pessimistic algorithm)

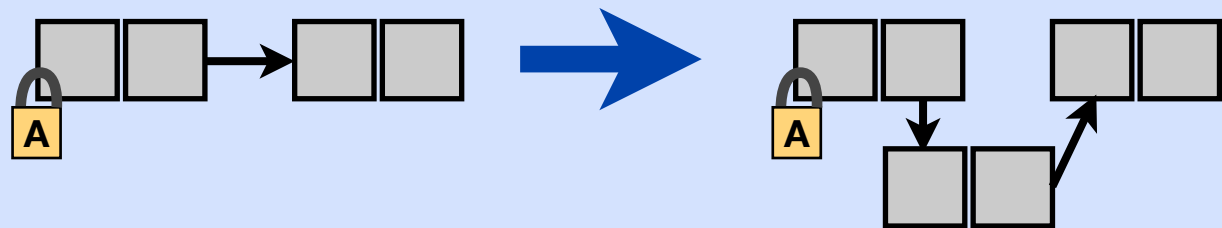
Lock



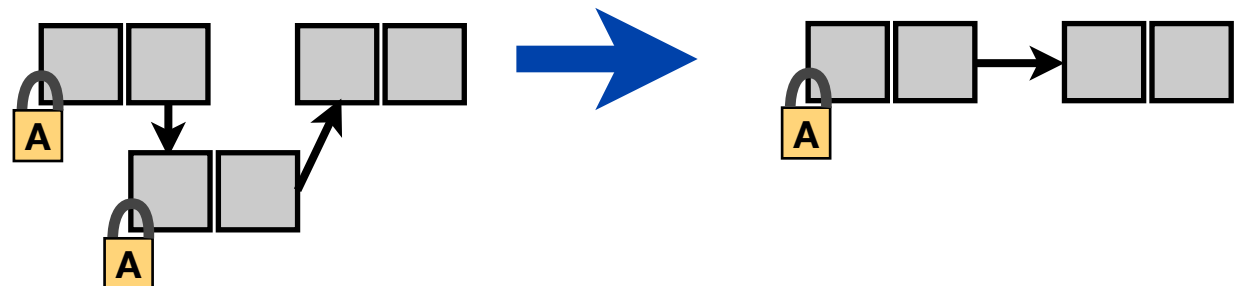
Unlock



Add node

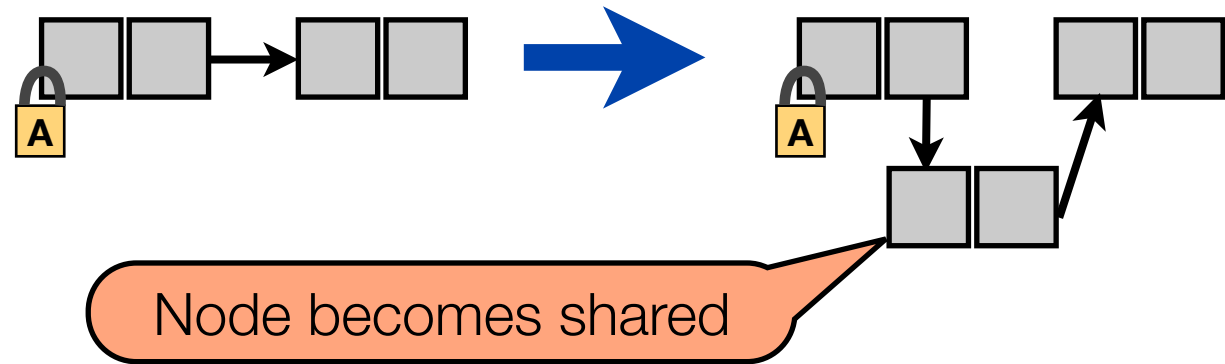


Delete node

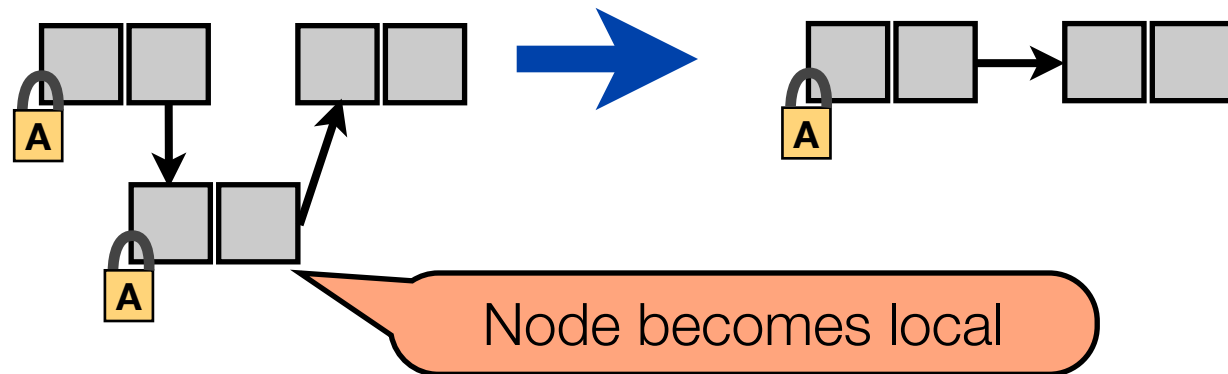


Ownership transfer

Add node

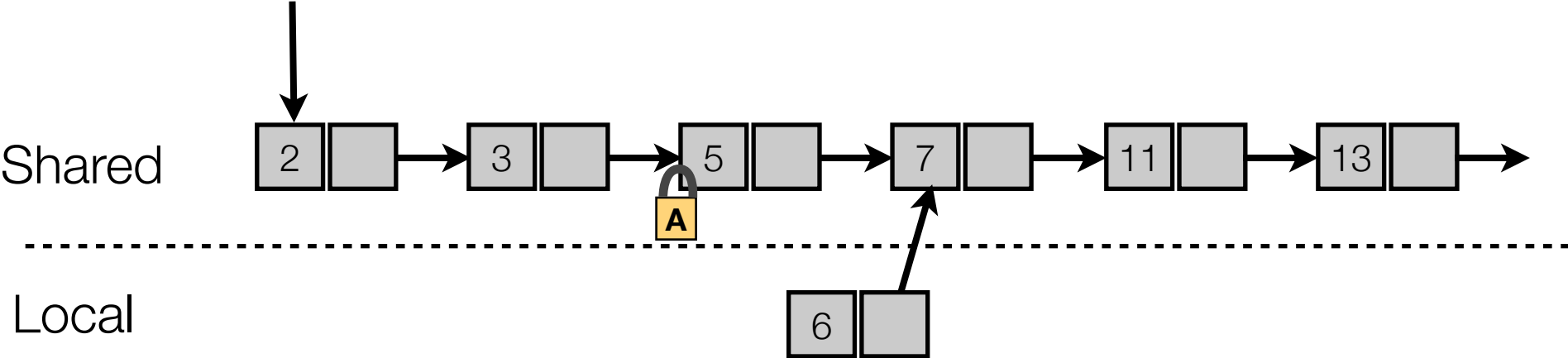


Delete node

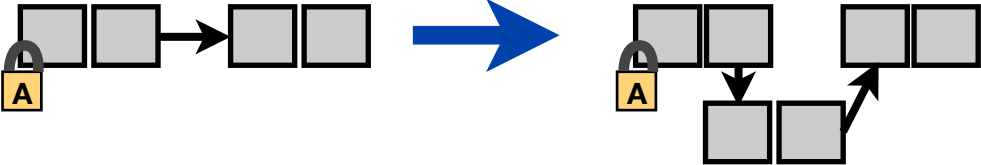


Local and shared state

Pessimistic algorithm

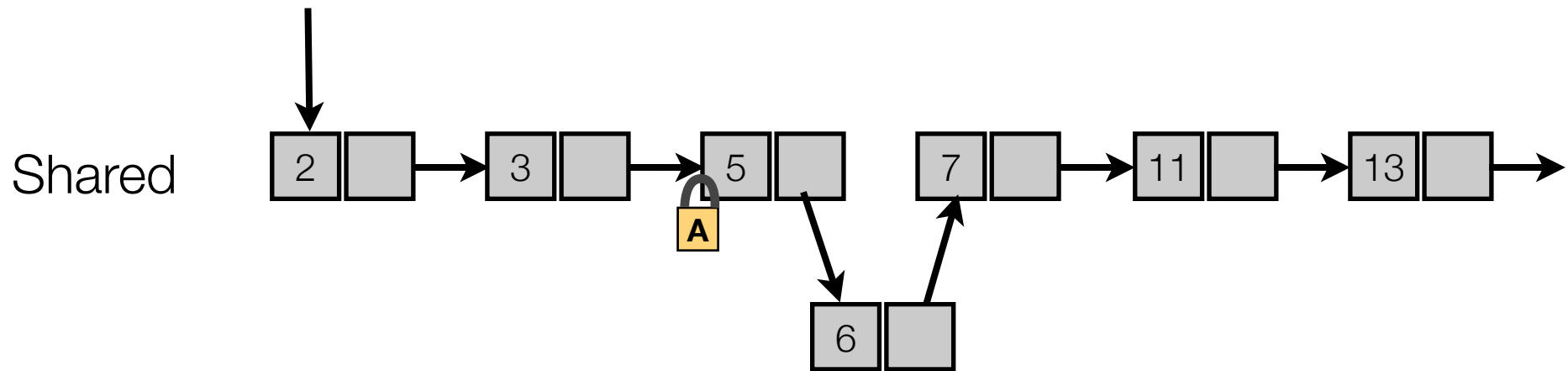


Add node



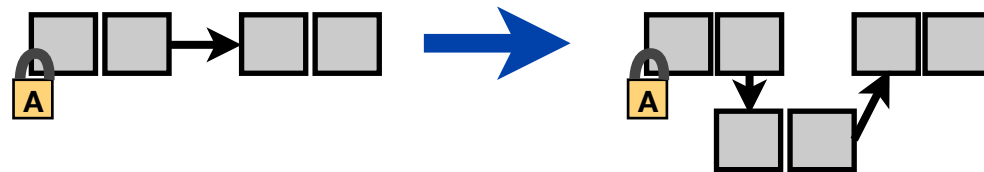
Local and shared state

Pessimistic algorithm



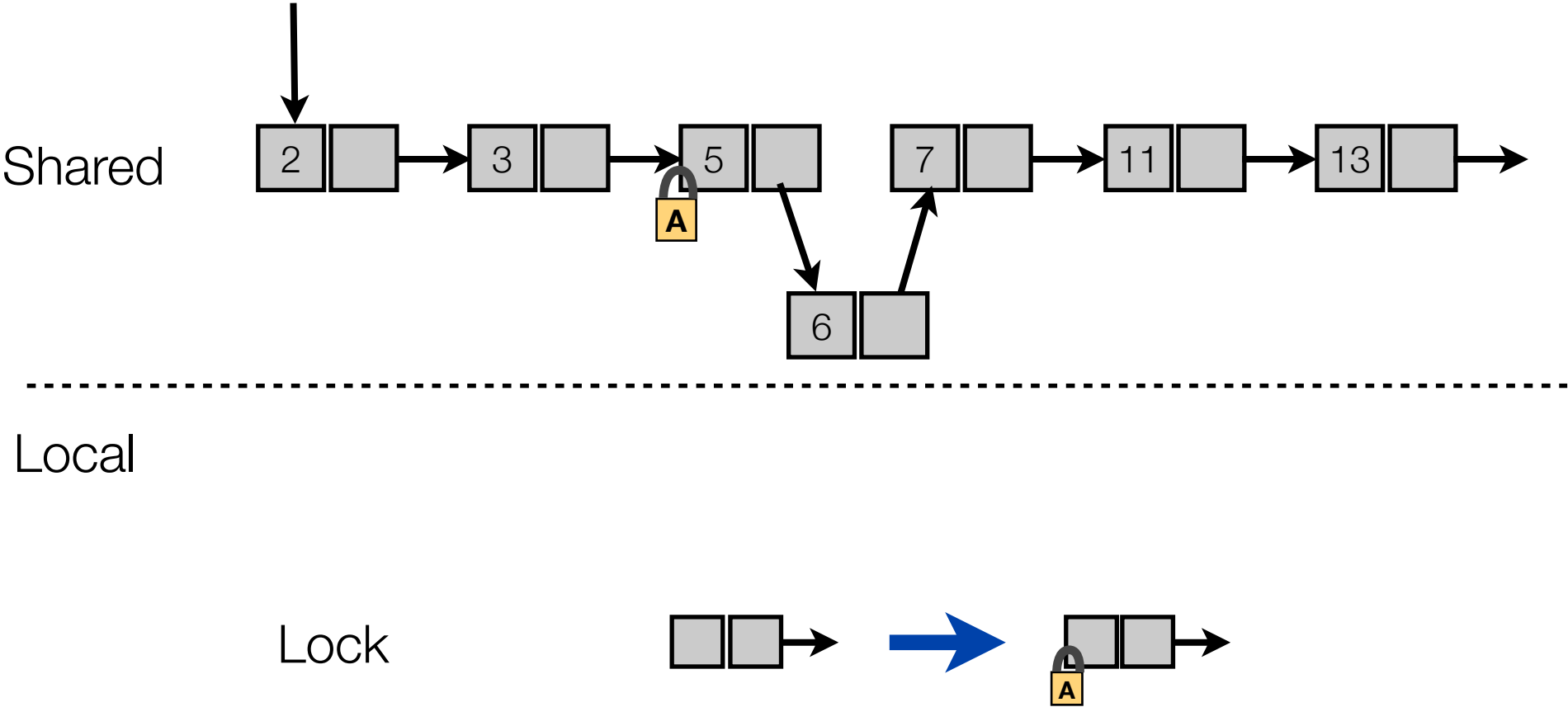
Local

Add node



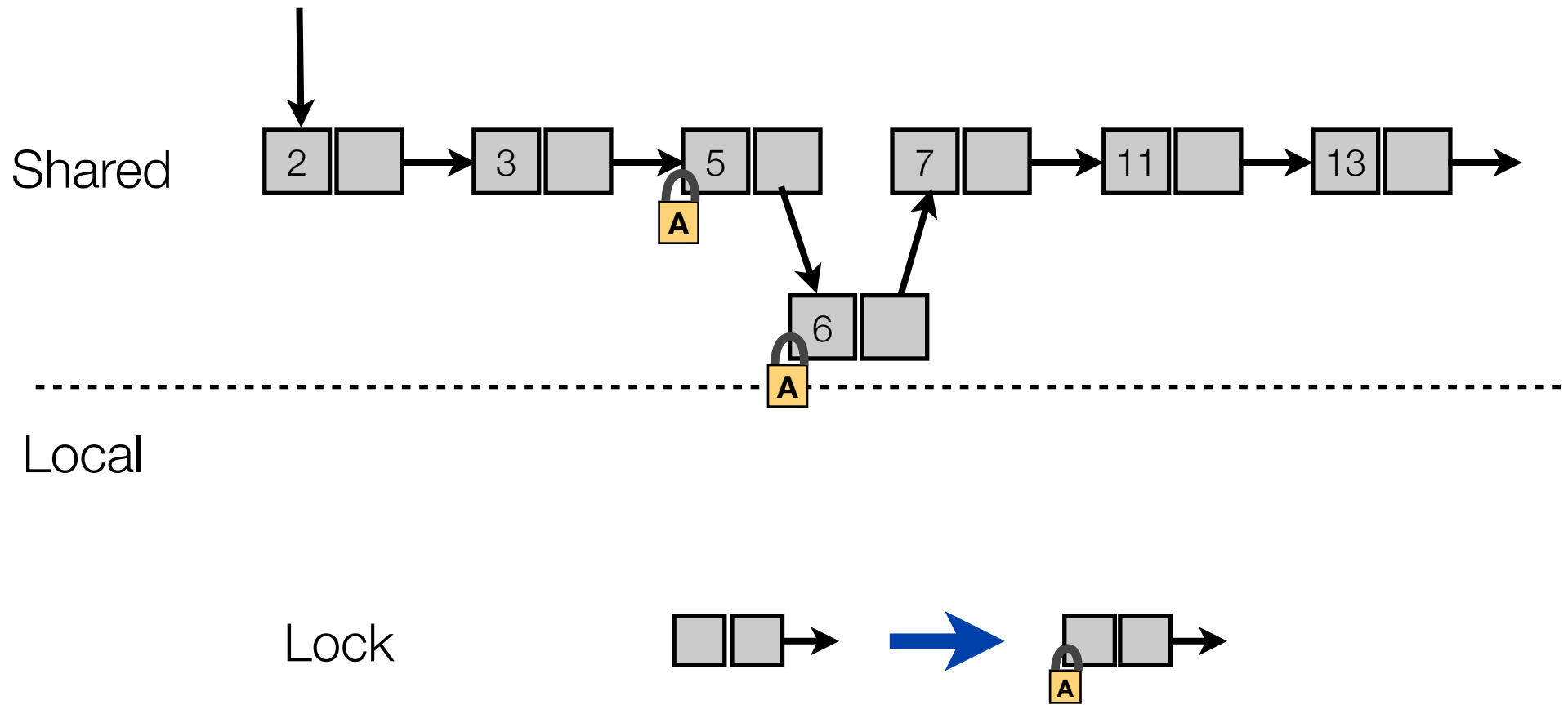
Local and shared state

Pessimistic algorithm



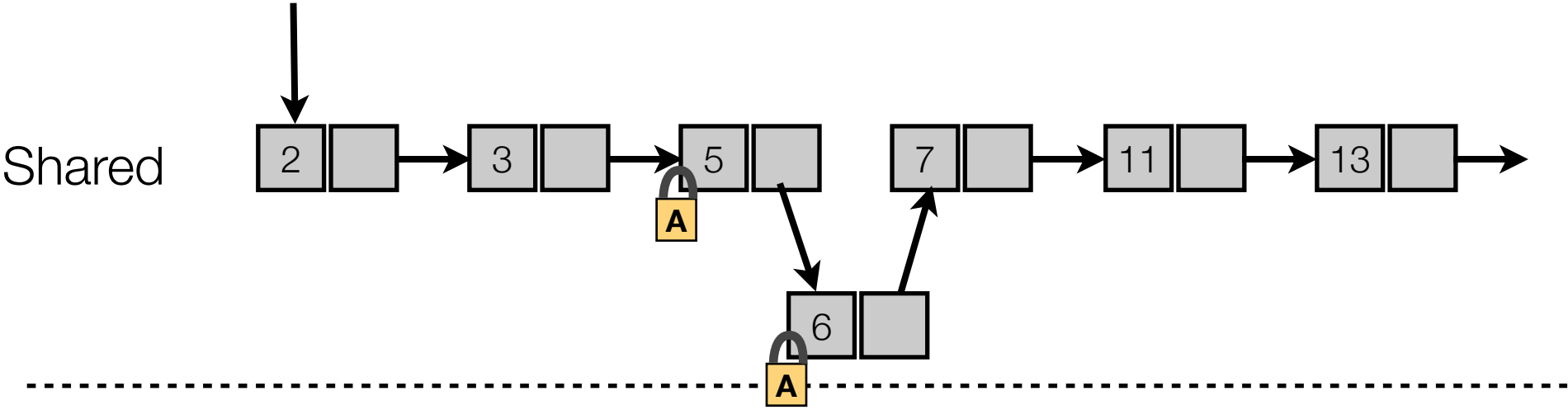
Local and shared state

Pessimistic algorithm

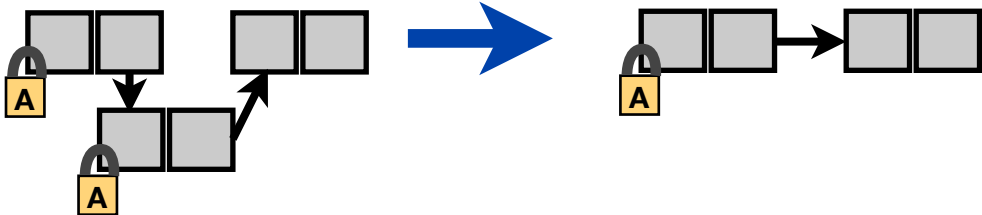


Local and shared state

Pessimistic algorithm

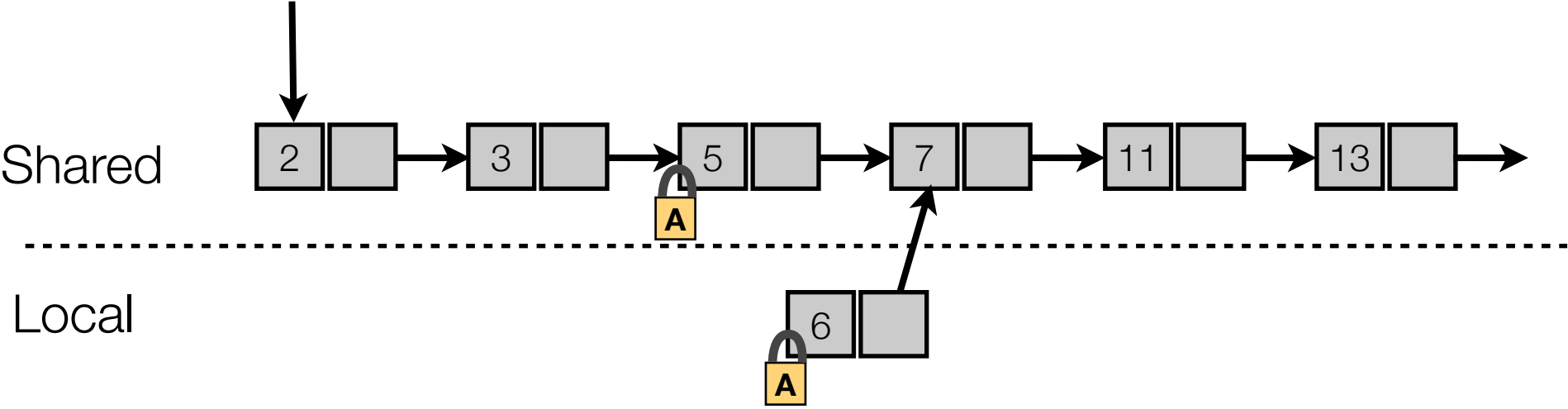


Delete node

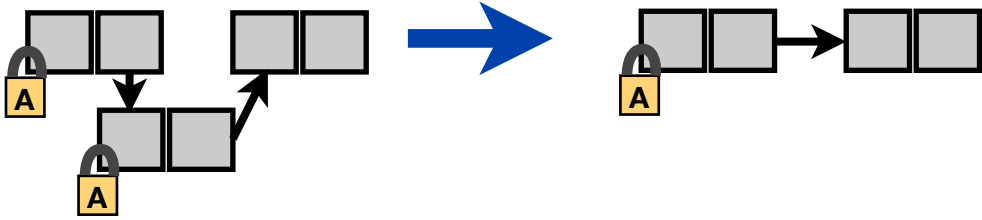


Local and shared state

Pessimistic algorithm

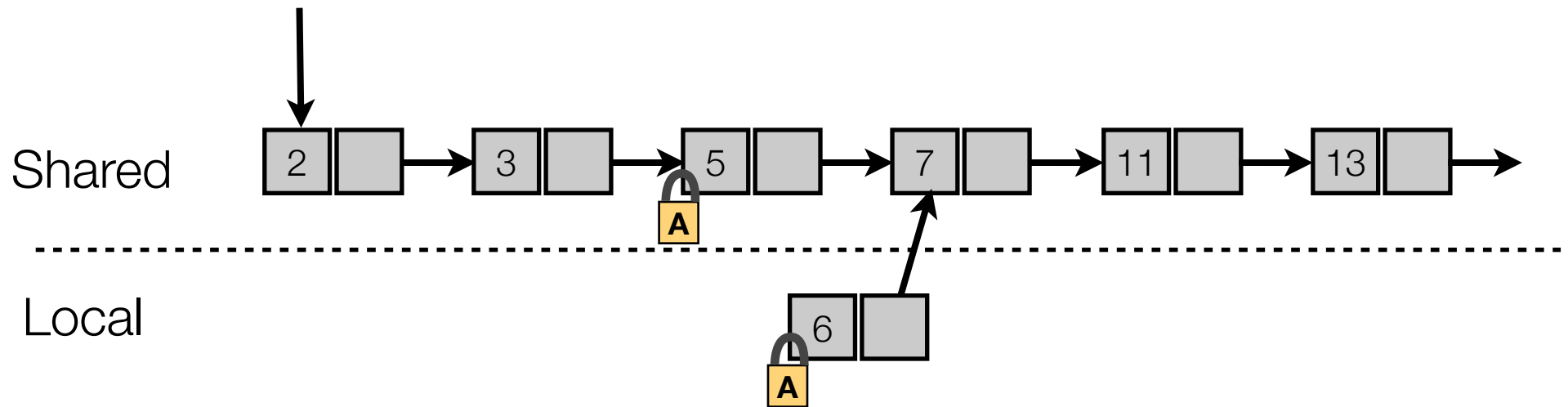


Delete node



Local and shared state

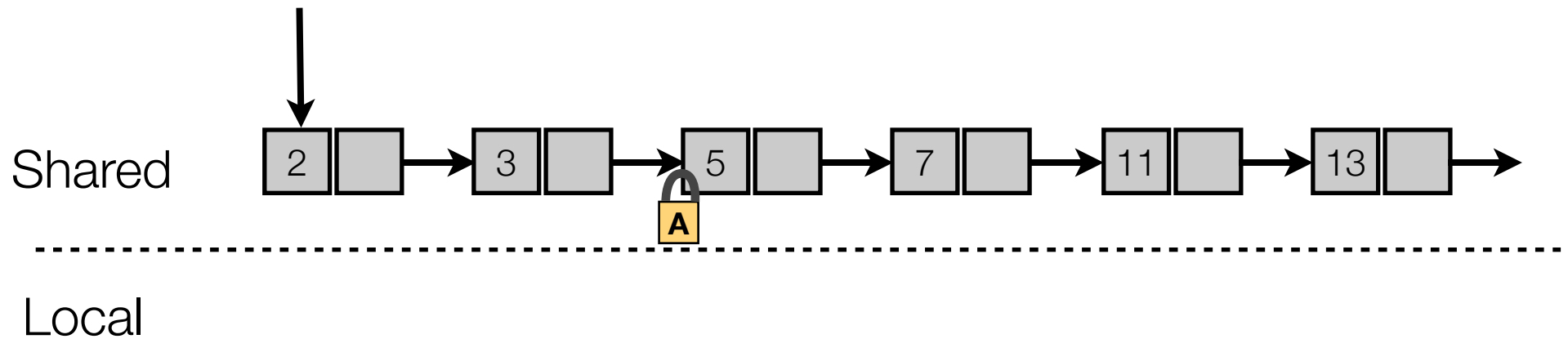
Pessimistic algorithm



Now, the node is local; we can safely dispose it.

Local and shared state

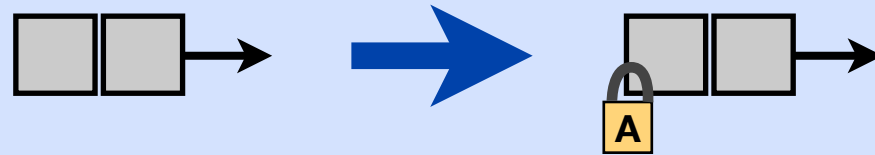
Pessimistic algorithm



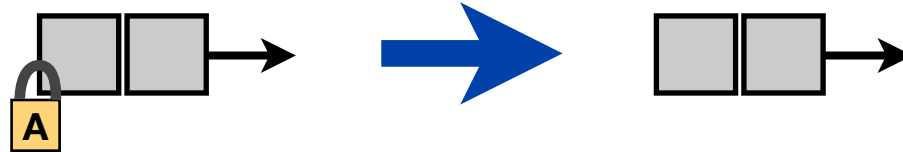
Now, the node is local; we can safely dispose it.

Actions (optimistic algorithm)

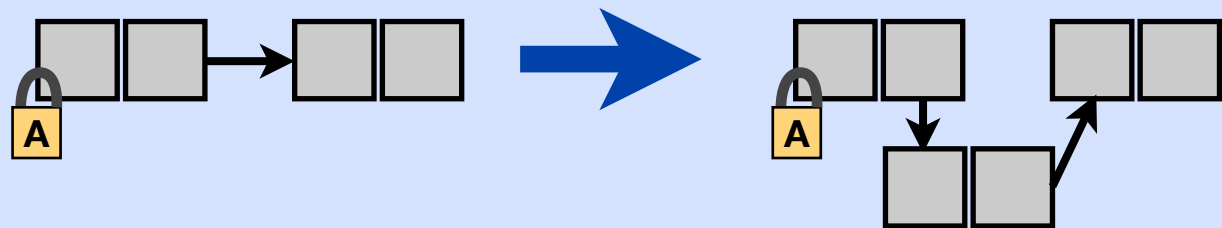
Lock



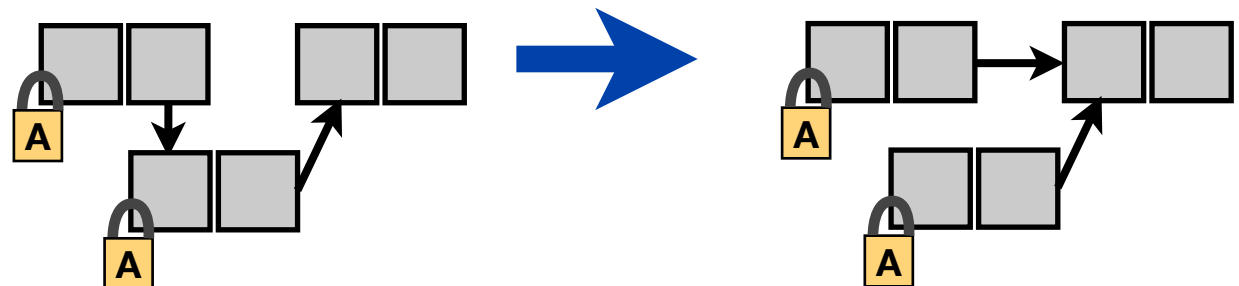
Unlock



Add node

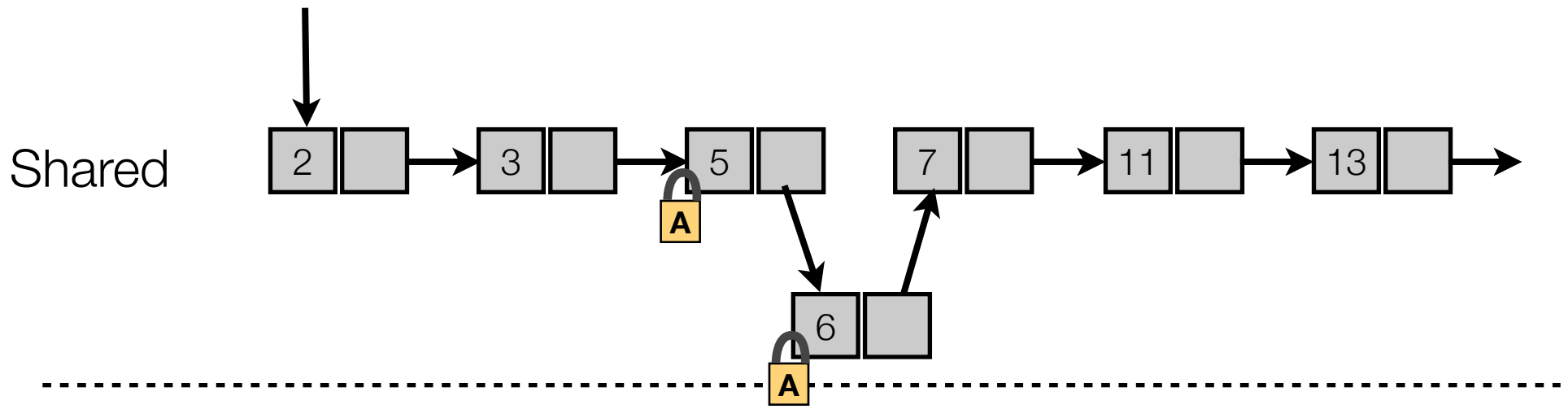


Delete node

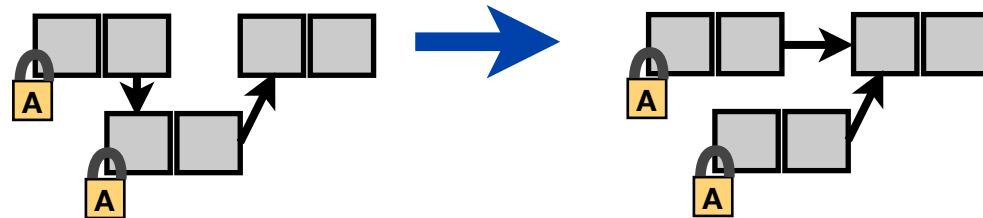


Local and shared state

Optimistic algorithm

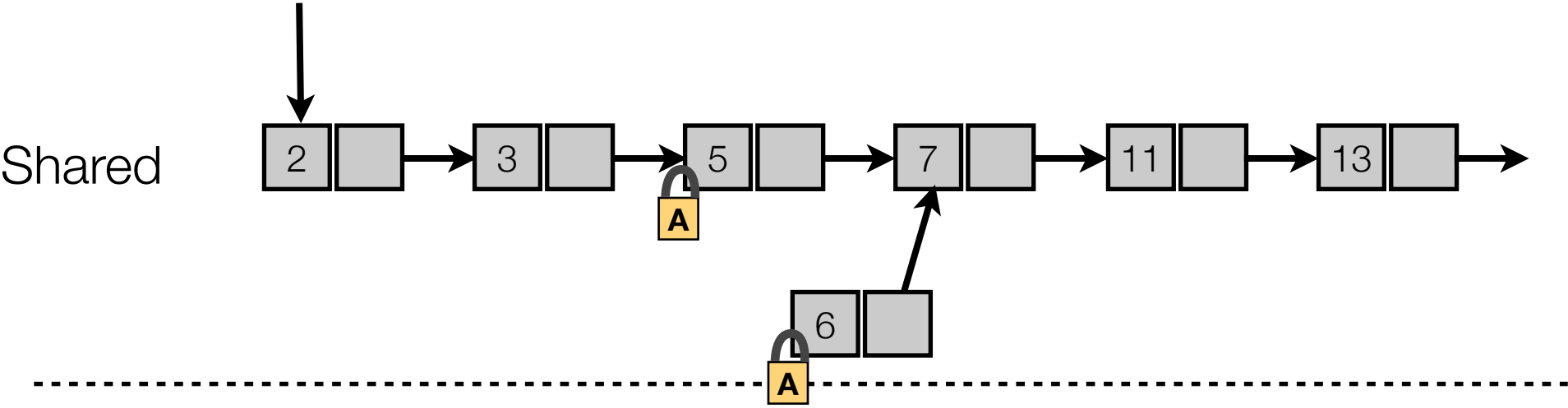


Delete node

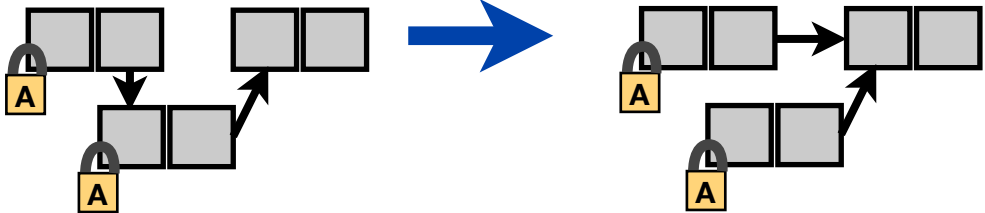


Local and shared state

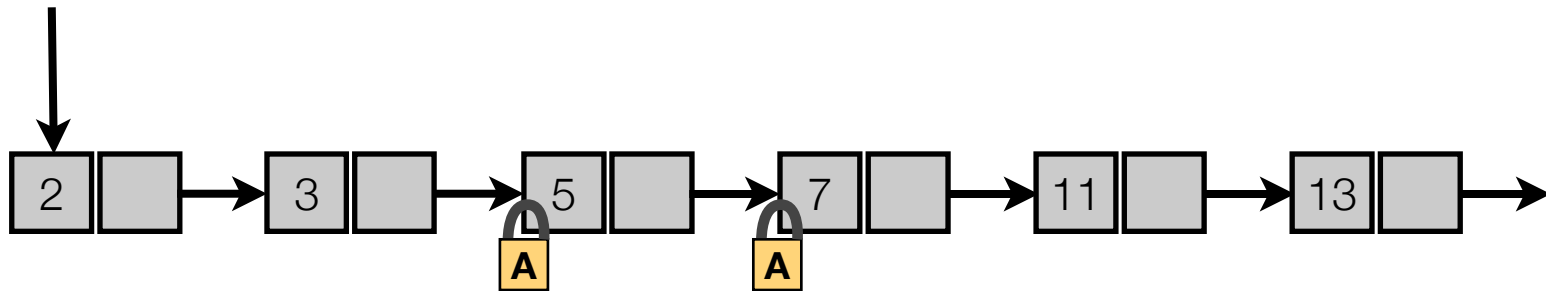
Optimistic algorithm



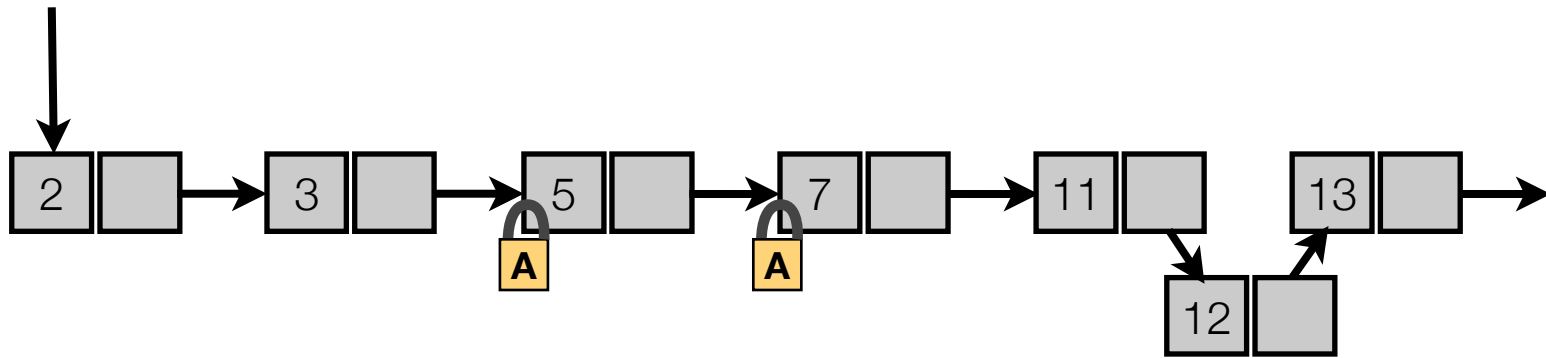
Delete node



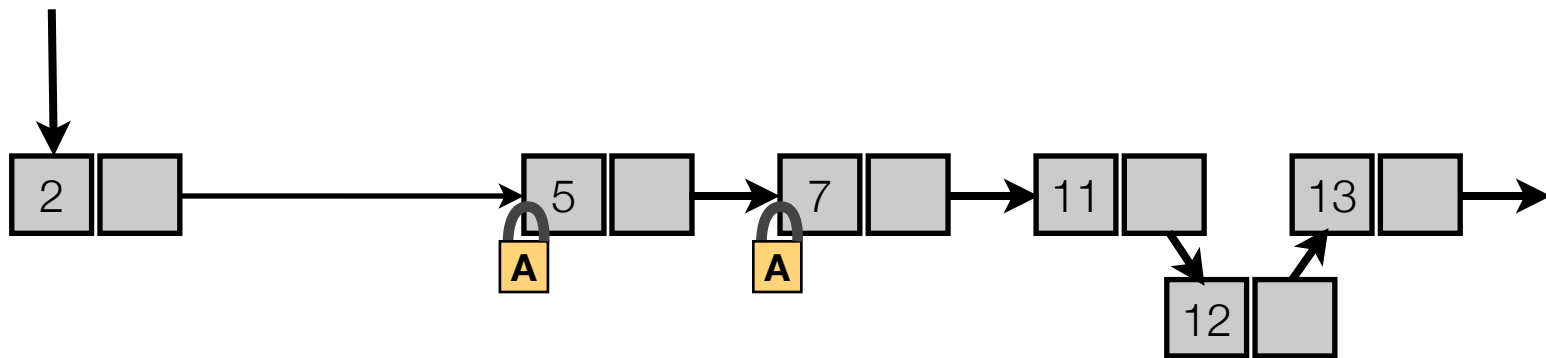
Interference: other threads



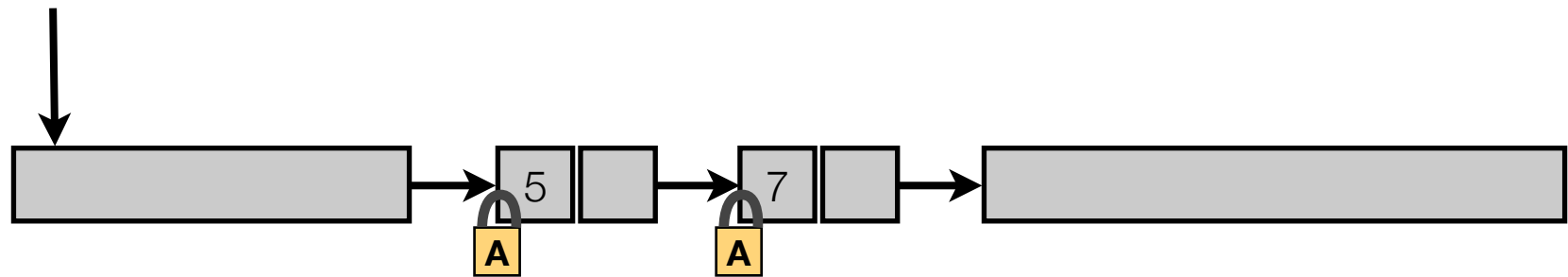
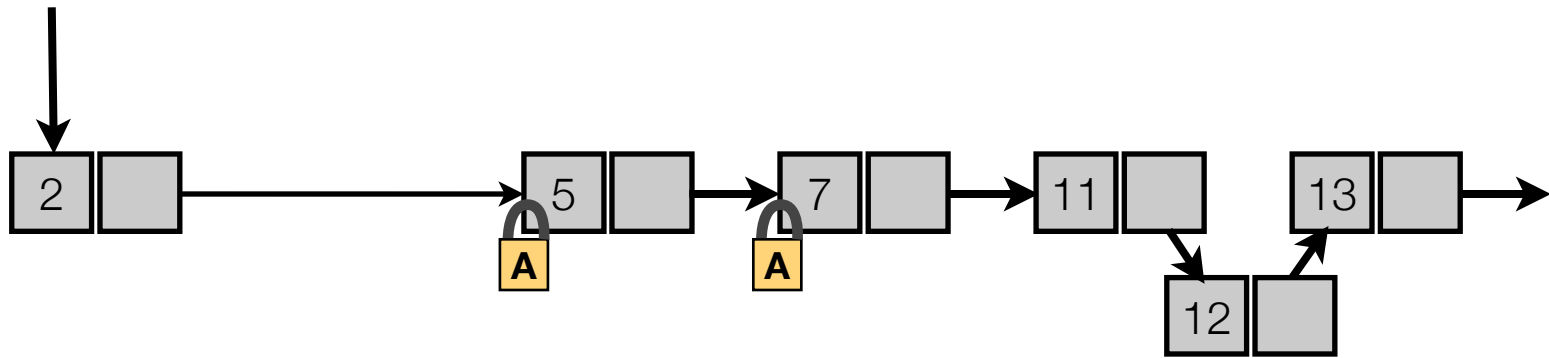
Interference: other threads



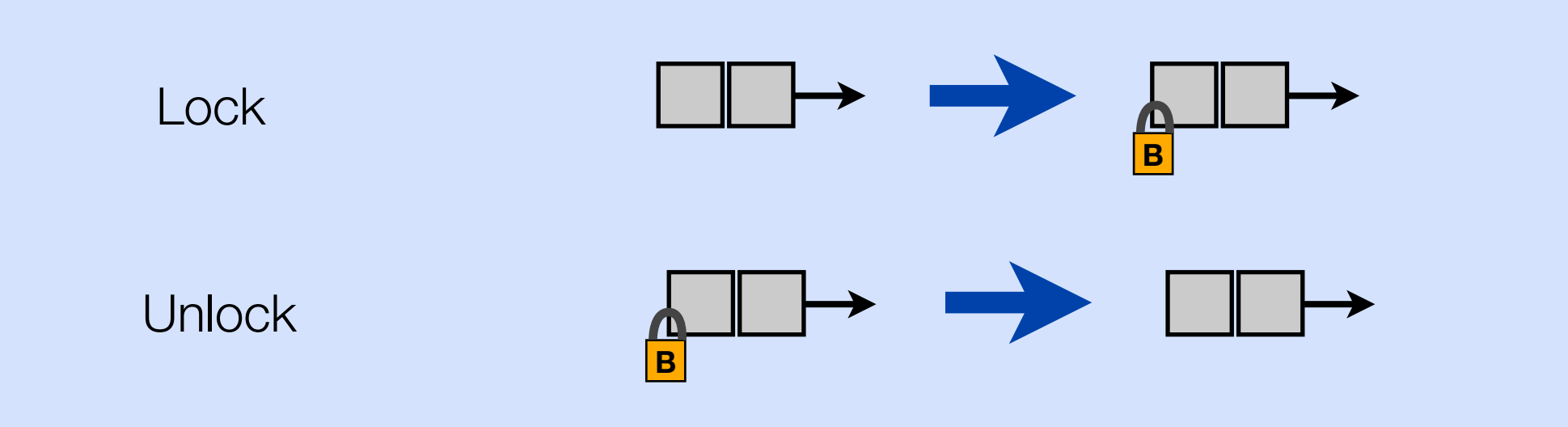
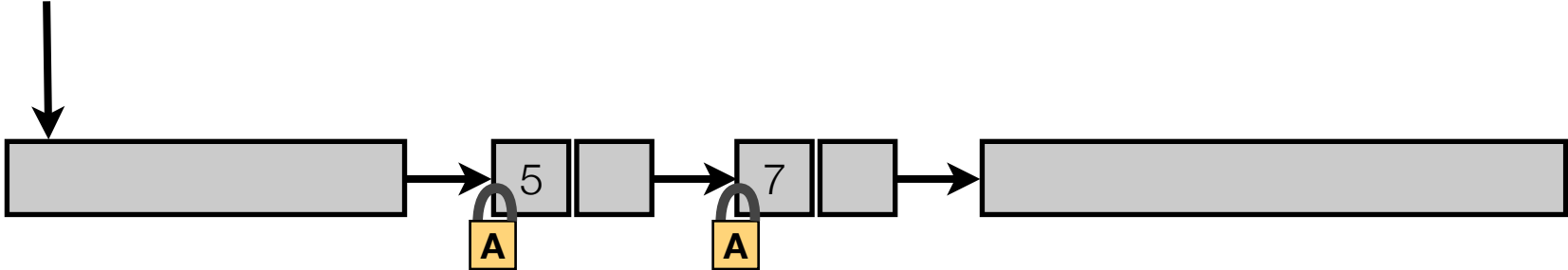
Interference: other threads



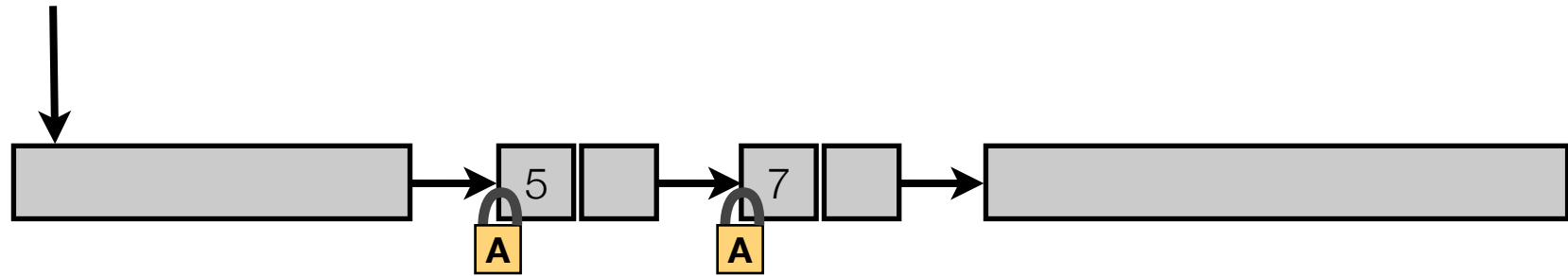
Interference: other threads



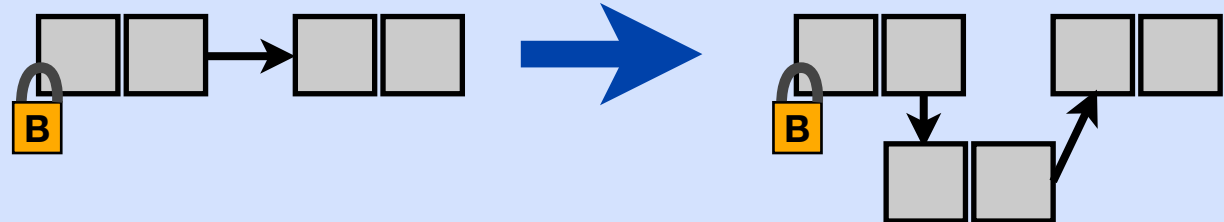
Stability



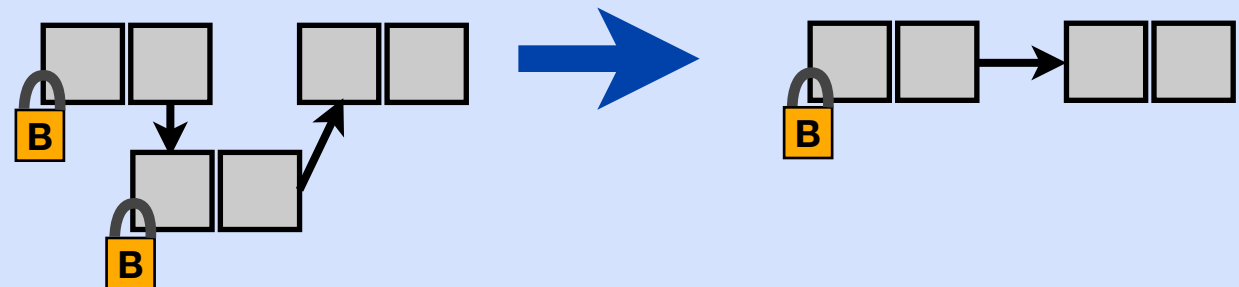
Stability



Add node



Delete node



Assertions

P, Q, R, \dots — separation logic assertions

p, q, r, \dots — RGSep assertions

shared state assertion

$p ::= P \mid \boxed{P} \mid p * q \mid p \vee q \mid p \wedge q \mid \exists x. p \mid \forall x. p$

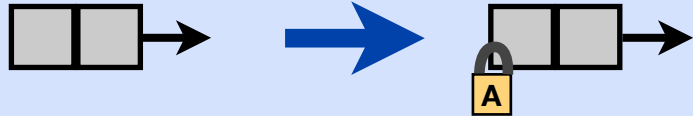
local state assertion

$$P(l, s) \stackrel{\text{def}}{\iff} P(l)$$

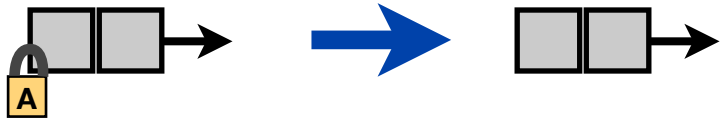
$$\boxed{P}(l, s) \stackrel{\text{def}}{\iff} P(s)$$

$$(p * q)(l, s) \stackrel{\text{def}}{\iff} \exists l_1 l_2. \text{dom}(l_1) \cap \text{dom}(l_2) = \emptyset \wedge l = l_1 \cup l_2 \wedge p_1(l_1, s) \wedge p_2(l_2, s)$$

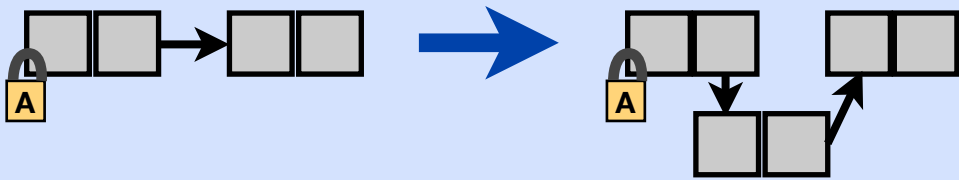
Actions



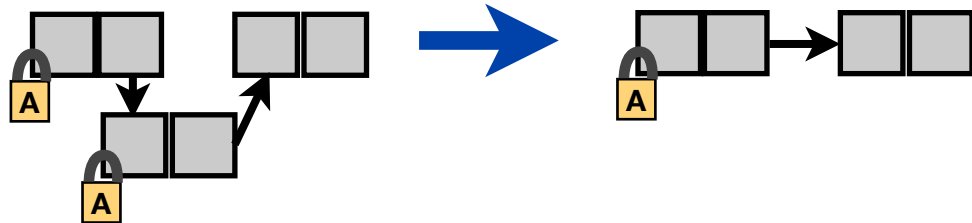
$x \mapsto 0, v, y \rightsquigarrow x \mapsto A, v, y$



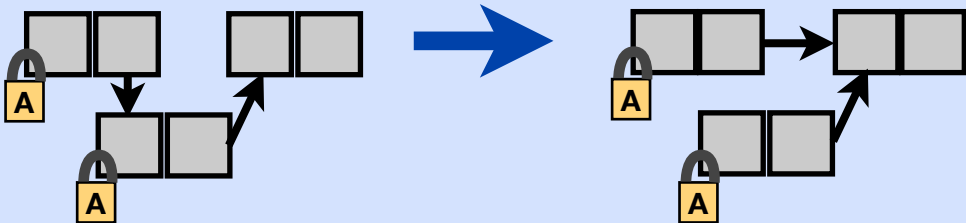
$x \mapsto A, v, y \rightsquigarrow x \mapsto 0, v, y$



$x \mapsto A, v, y \rightsquigarrow$
 $x \mapsto A, v, z$
 $* z \mapsto 0, w, y$



$x \mapsto A, v, y \rightsquigarrow x \mapsto A, v, z$
 $* y \mapsto A, w, z$



$x \mapsto A, v, y \rightsquigarrow x \mapsto A, v, z$
 $* y \mapsto A, w, z \rightsquigarrow * y \mapsto A, w, z$

Parallel composition

$C_1 \text{ sat } (p_1, R \cup G_2, G_1, q_1)$

$C_2 \text{ sat } (p_2, R \cup G_1, G_2, q_2)$

$(C_1 \parallel C_2) \text{ sat } (p_1 * p_2, R, G_1 \cup G_2, q_1 * q_2)$

Atomic commands

$$\frac{\{P\} C \{Q\}}{C \text{ sat } (P, R, G, Q)} \quad \text{Local commands}$$

$$\frac{(P_2, Q_2) \in G \quad \{P_1 * P_2\} C \{Q_1 * Q_2\}}{(\text{atomic } C) \text{ sat } (P_1 * \boxed{P_2 * F}, \emptyset, G, Q_1 * \boxed{Q_2 * F})}$$

$$\frac{p, q \text{ stable under } R \quad (\text{atomic } C) \text{ sat } (p, \emptyset, G, q)}{(\text{atomic } C) \text{ sat } (p, R, G, q)}$$

Stability

- Local state assertions are trivially stable
- Shared state assertions:

\boxed{S} is stable under (P, Q)

if and only if

$$(P \text{ } \textcircled{*} \text{ } S) * Q \Rightarrow S$$

$$(P \text{ } \textcircled{*} \text{ } S)(h) \stackrel{\text{def}}{\iff} \exists h'. \text{dom}(h) \cap \text{dom}(h') = \emptyset \wedge P(h') \wedge S(h \cup h')$$

Some further topics

Tool support:

- Symbolic execution with stabilization
- Action inference
- Linearization point inference (SmallfootRG & Cave)

Deny-guarantee & concurrent abstract predicates:

- Make interference specs first class
- Logical/abstract separation