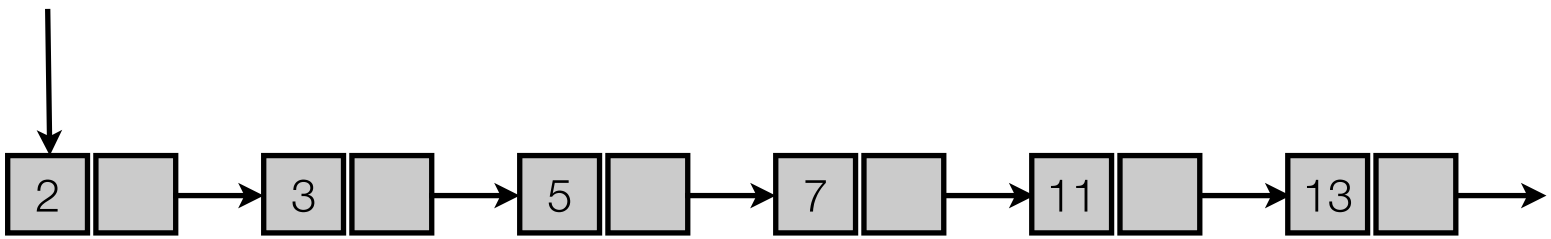
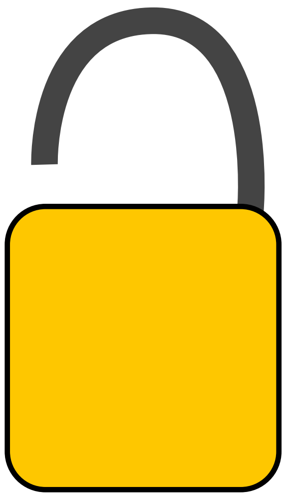
RGSep

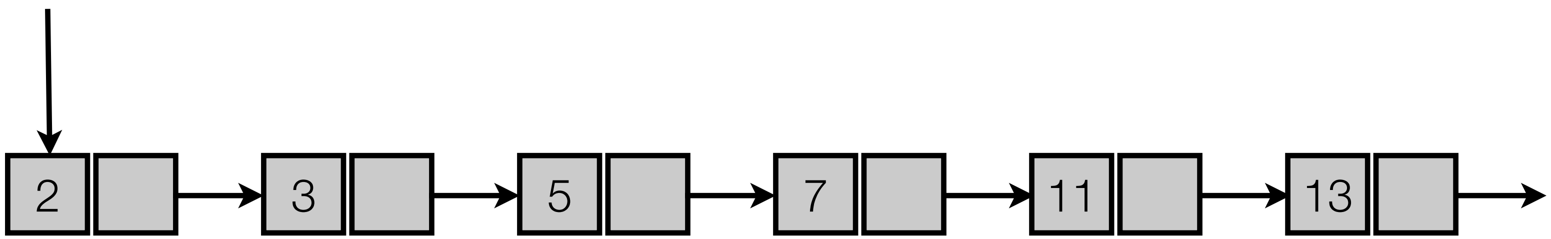
Viktor Vafeiadis

Coarse-grain locking





Fine-grain locking (pessimistic)



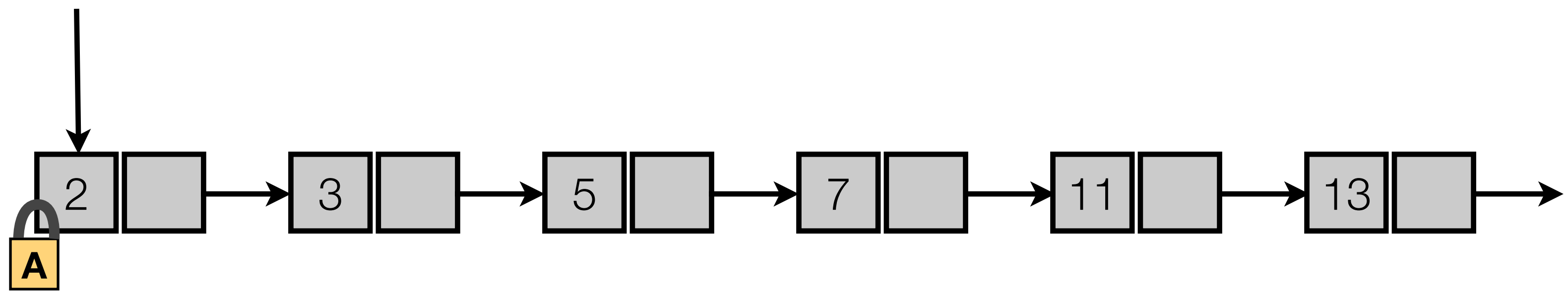
One lock per node:

— 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)



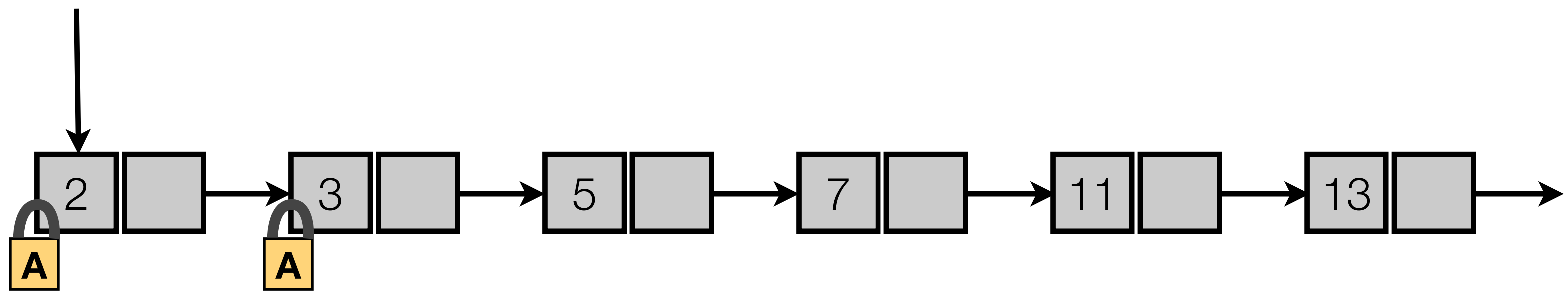
One lock per node:

— 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)



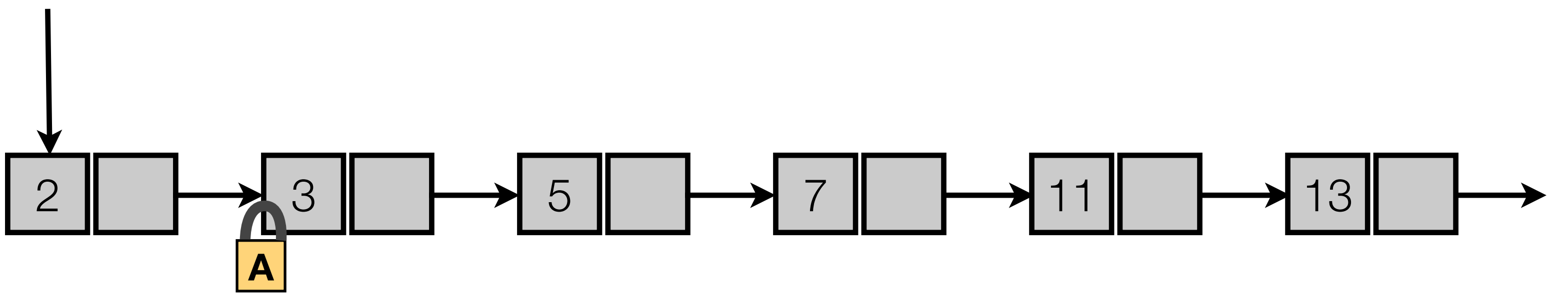
One lock per node:

— 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)



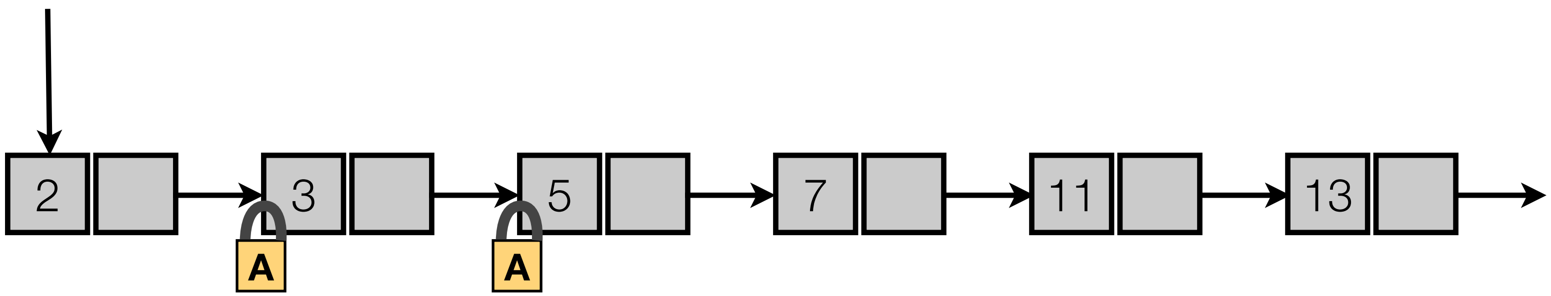
One lock per node:

— 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)



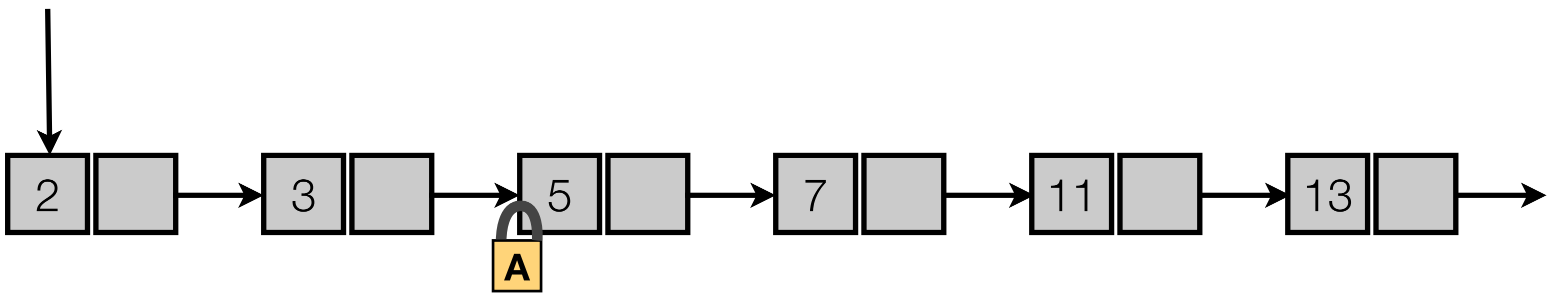
One lock per node:

— 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)



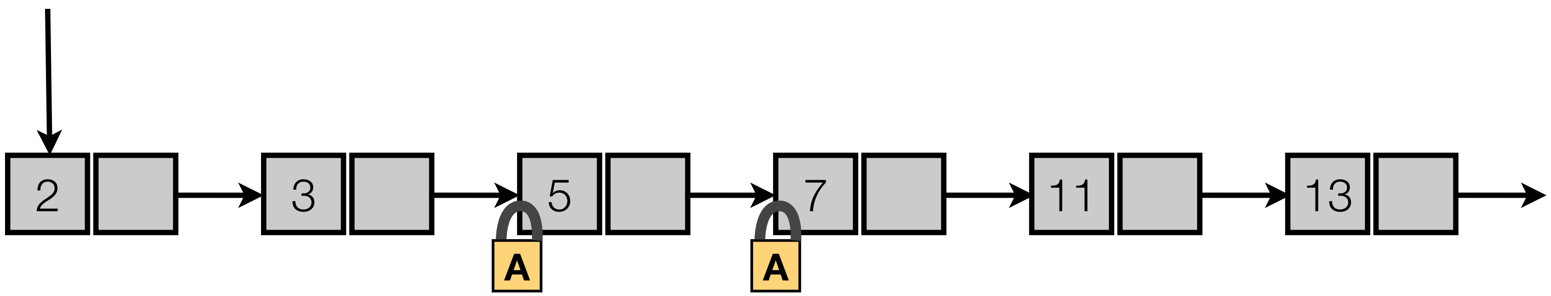
One lock per node:

— 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)



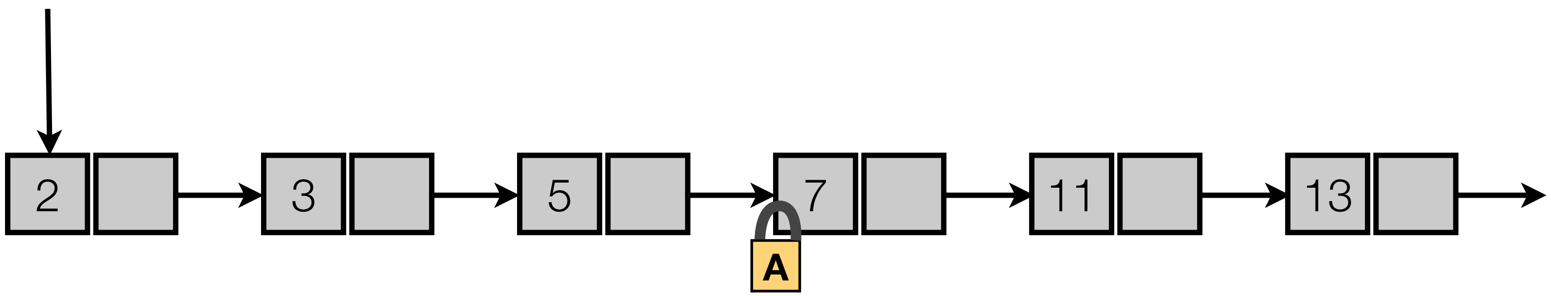
One lock per node:

— 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)



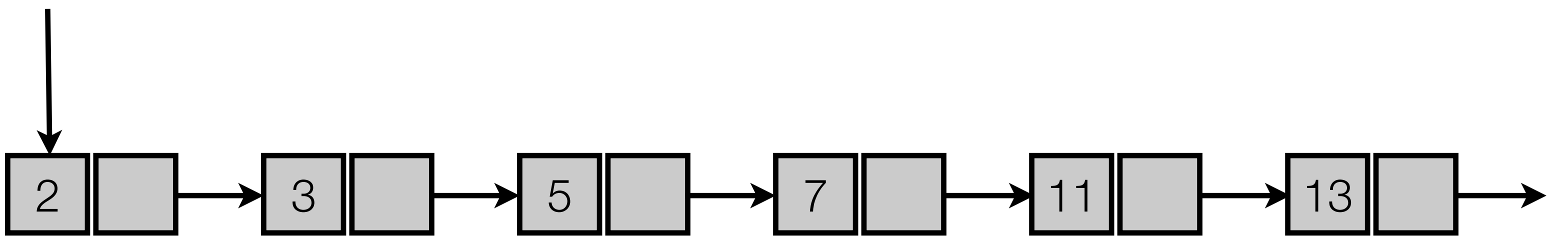
One lock per node:

— Traversals acquire locks in a “hand over hand” fashion.

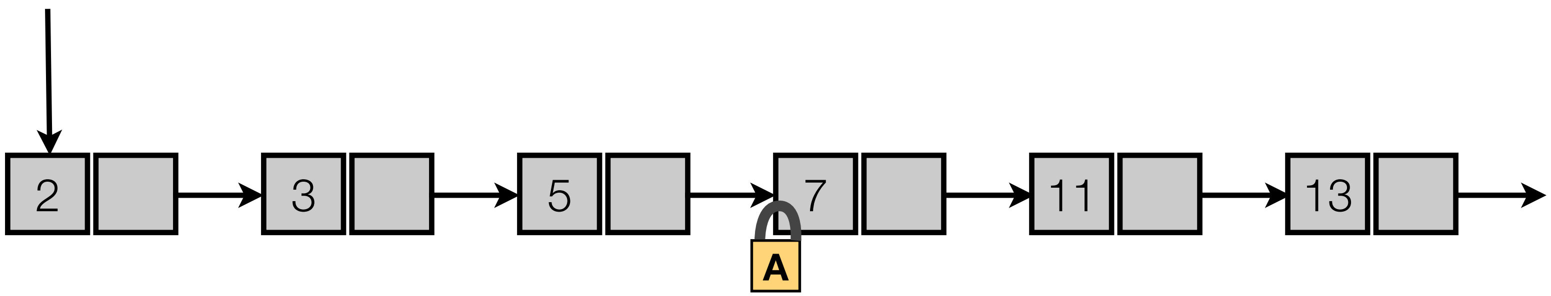
— If node is locked, we can add a node after it.

— If two adjacent nodes are locked, we can delete the second.

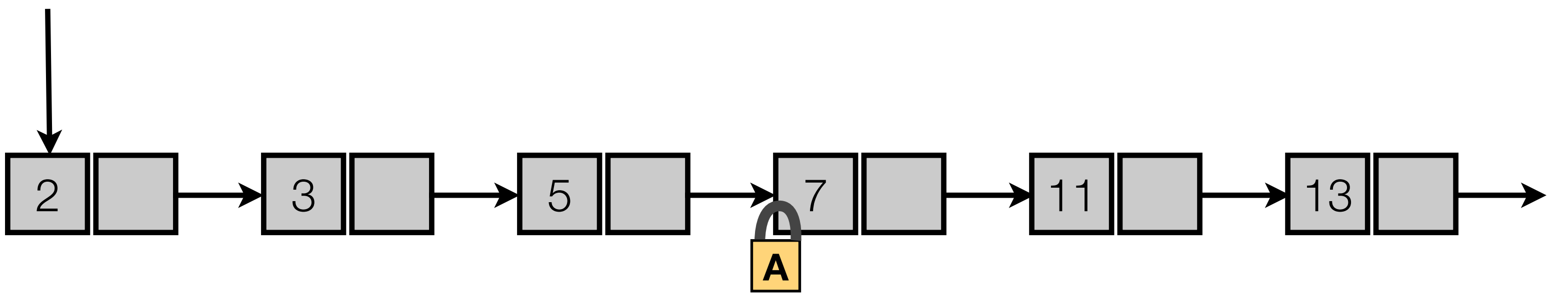
Fine-grain locking (optimistic)



Fine-grain locking (optimistic)



Fine-grain locking (optimistic)

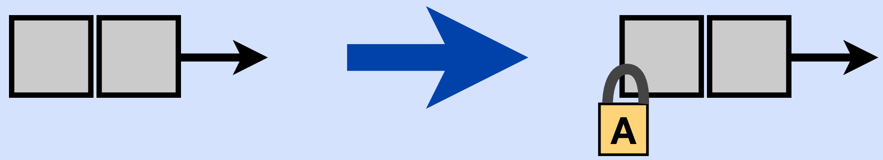


Re-traverse the list OR perform deletions in two steps

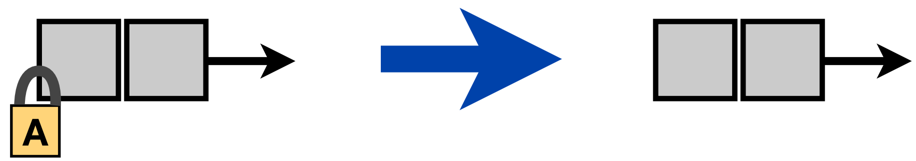
Leaks memory: cannot dispose deleted nodes.

Actions (pessimistic algorithm)

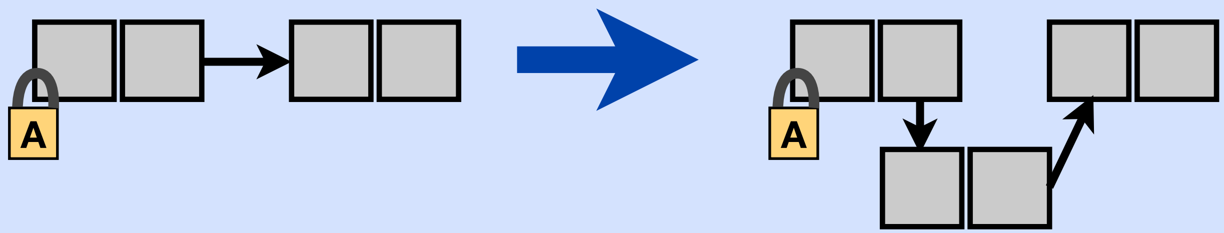
Lock



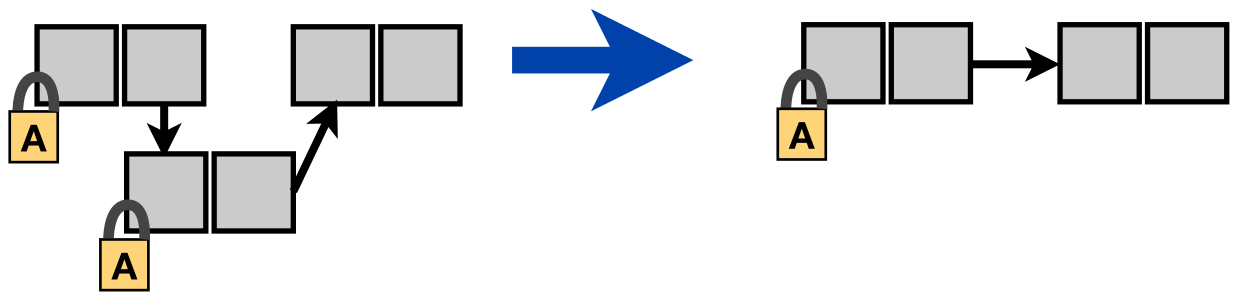
Unlock



Add node

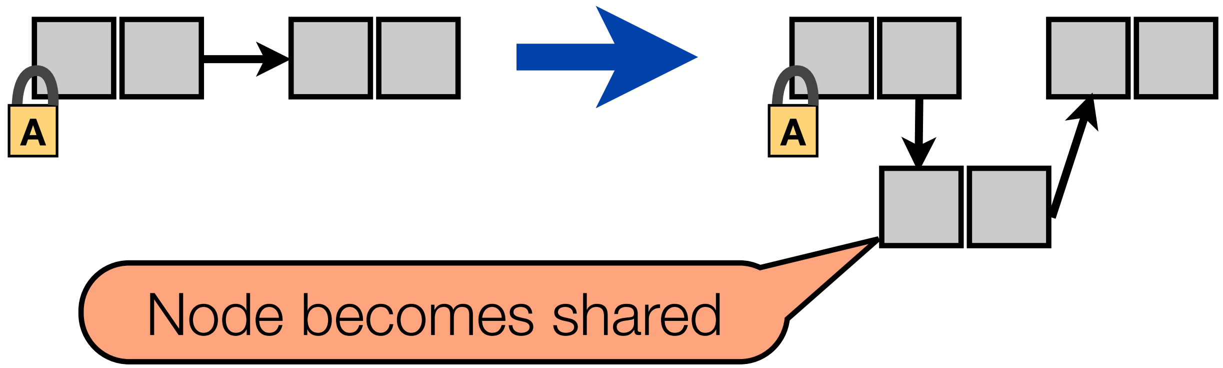


Delete node

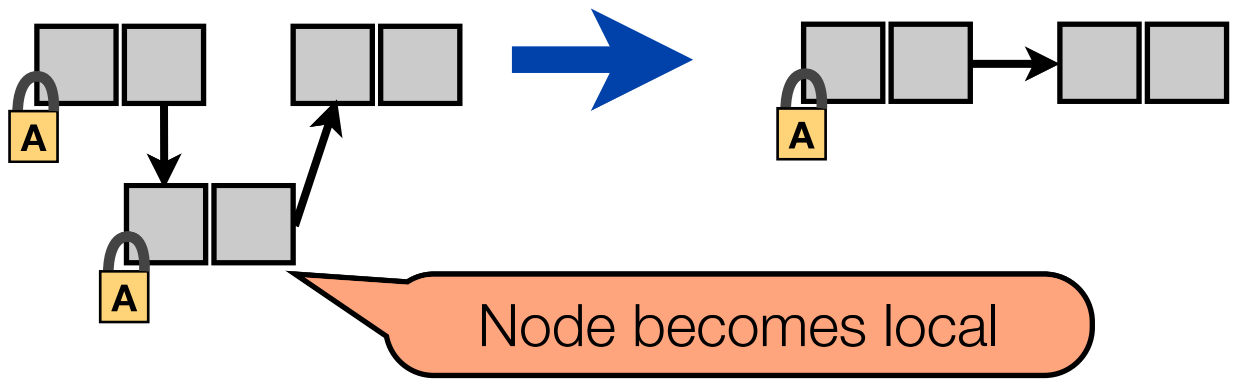


Ownership transfer

Add node

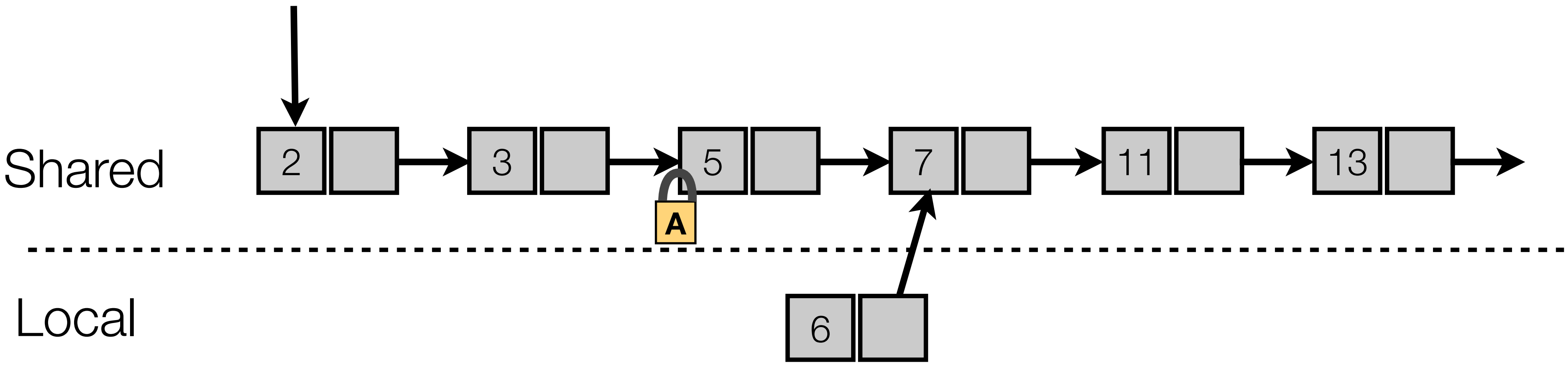


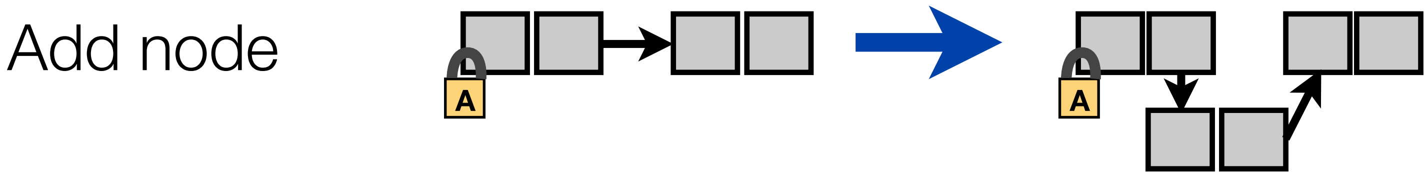
Delete node



Local and shared state

Pessimistic algorithm

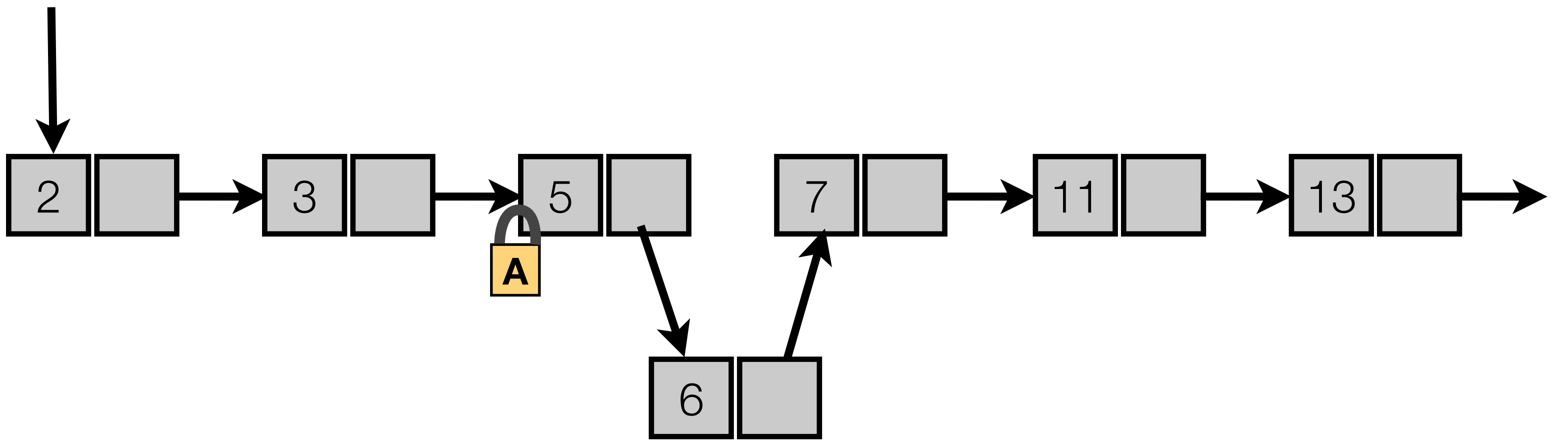




Local and shared state

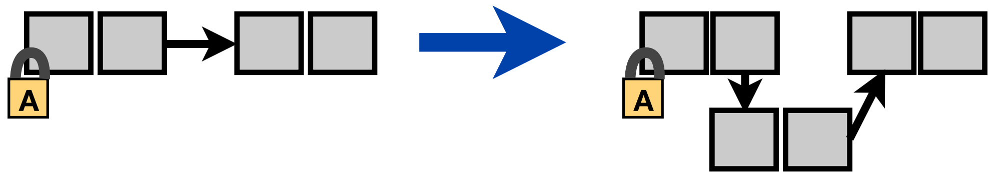
Pessimistic algorithm

Shared



Local

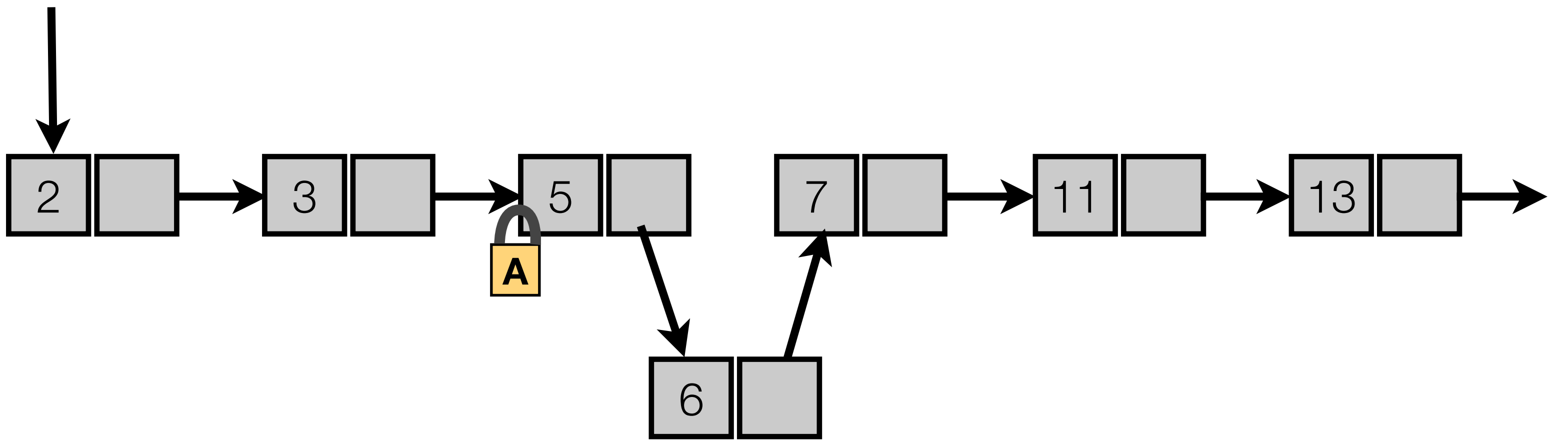
Add node



Local and shared state

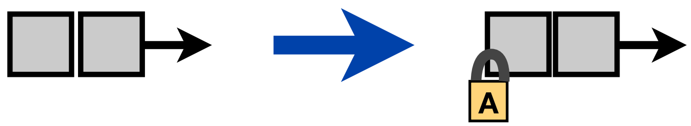
Pessimistic algorithm

Shared

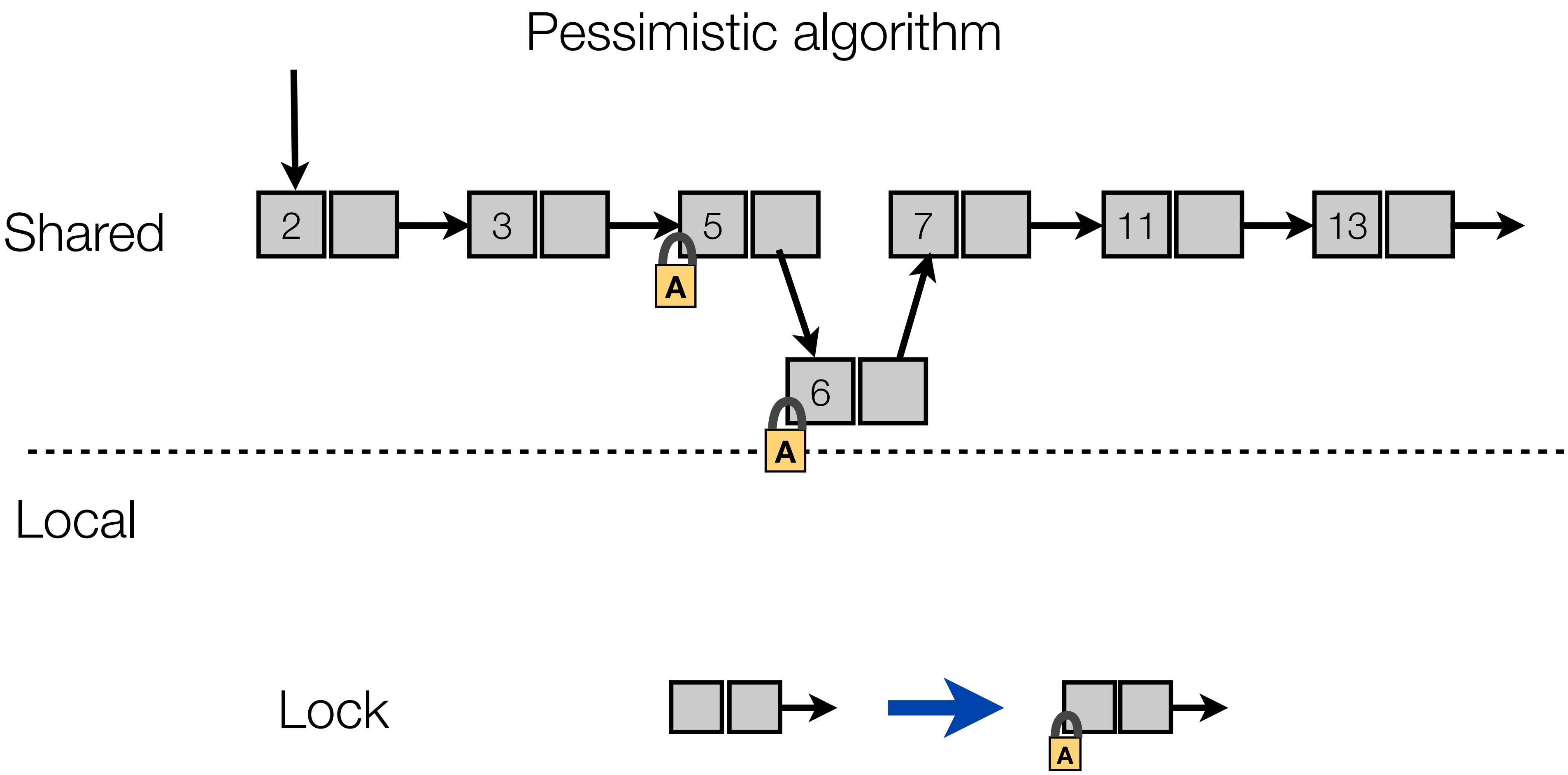


Local

Lock

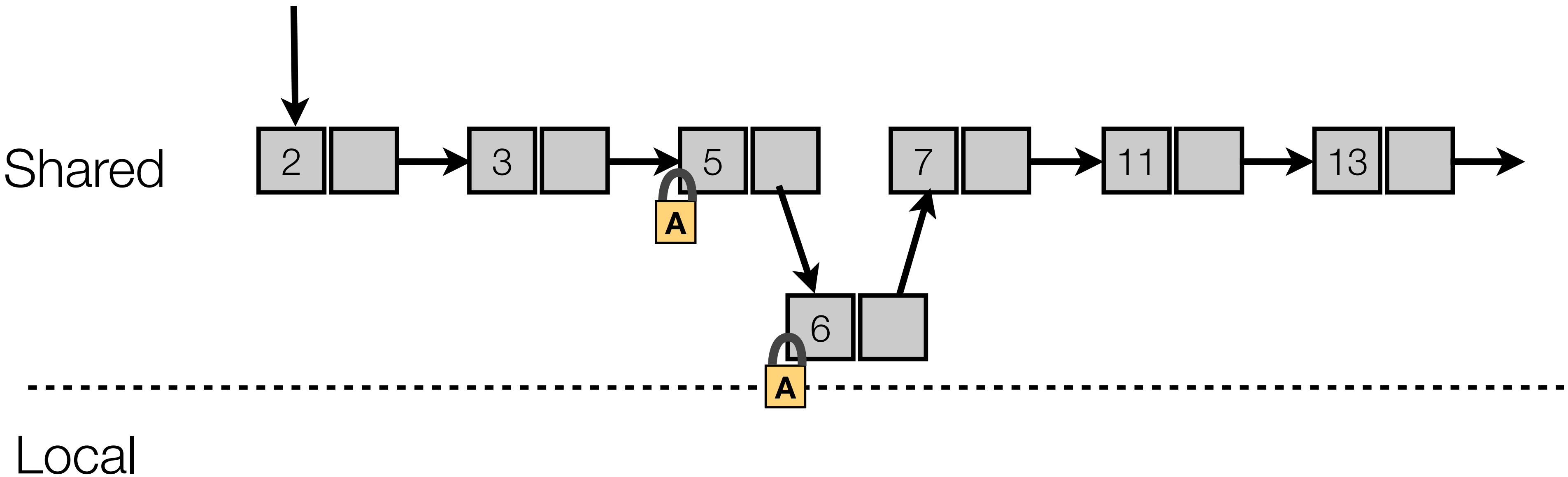


Local and shared state

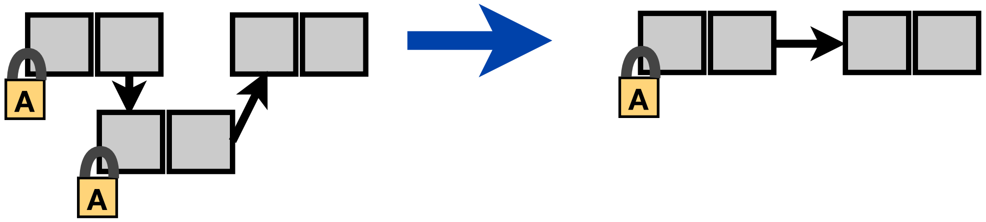


Local and shared state

Pessimistic algorithm

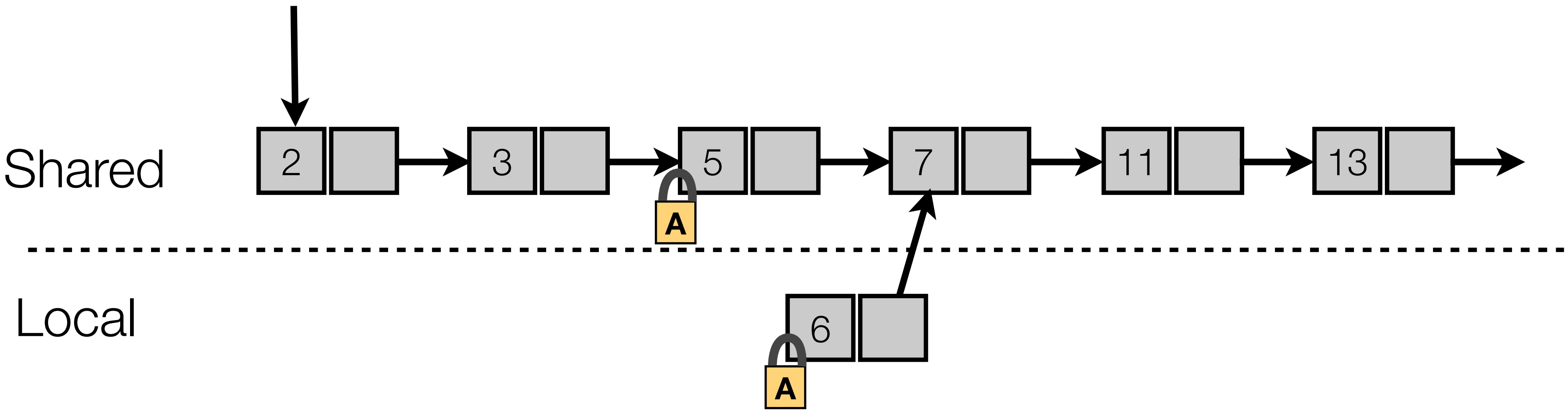


Delete node

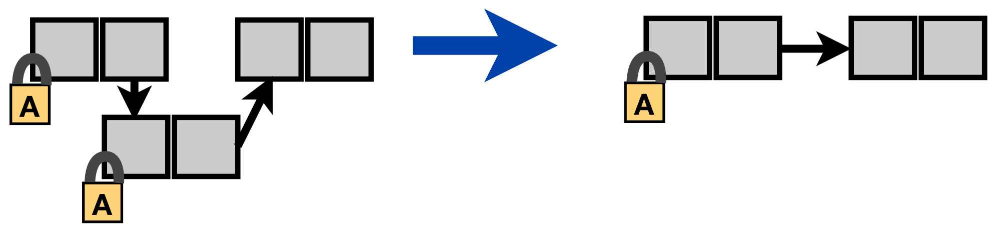


Local and shared state

Pessimistic algorithm

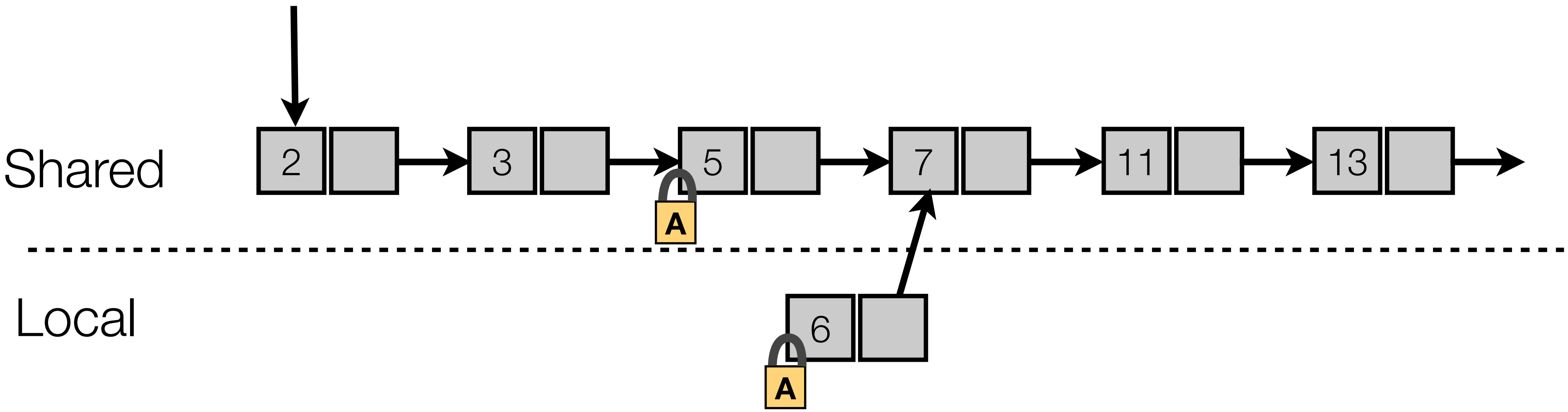


Delete node



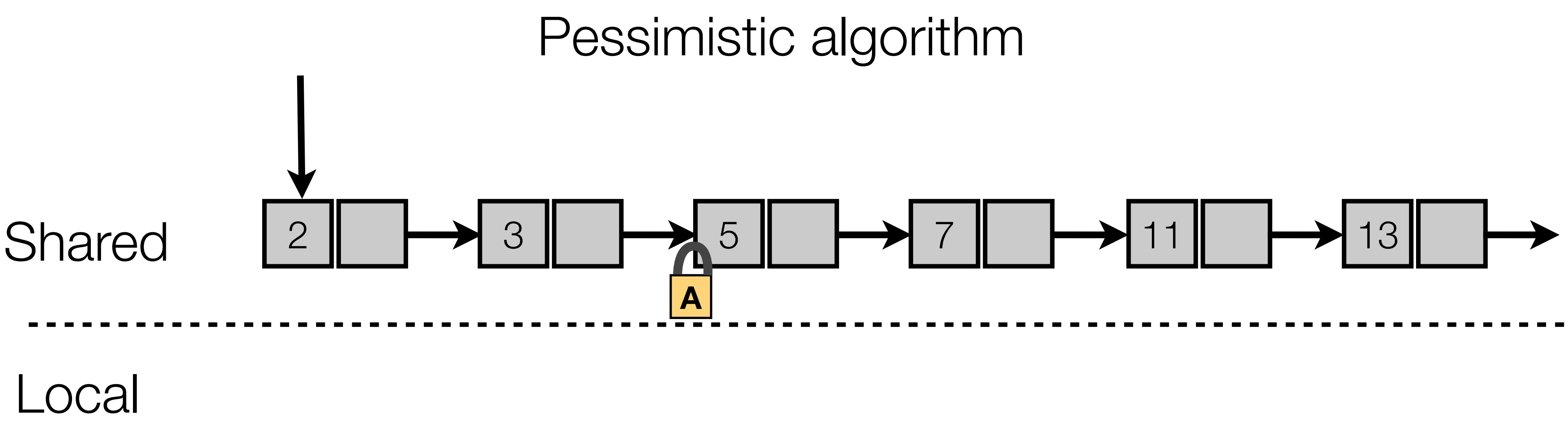
Local and shared state

Pessimistic algorithm



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

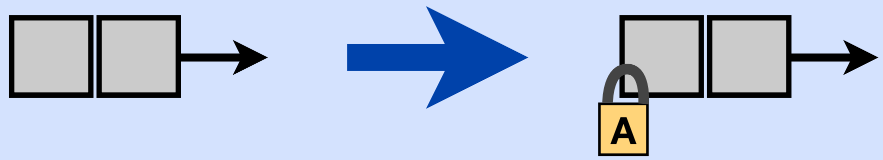
Local and shared state



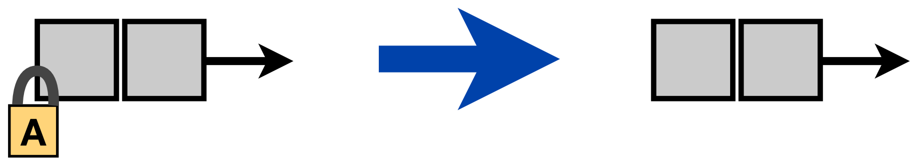
Now, the node is local; we can safely dispose it.

Actions (optimistic algorithm)

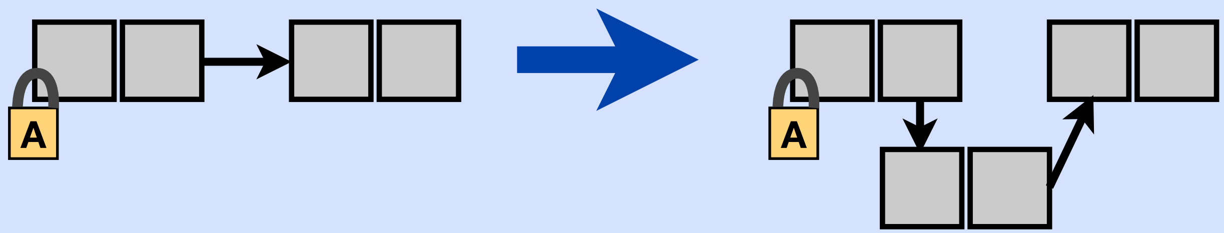
Lock



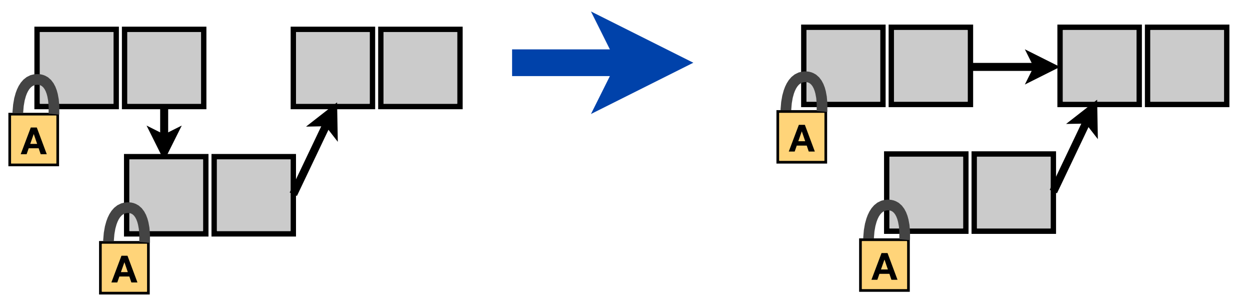
Unlock



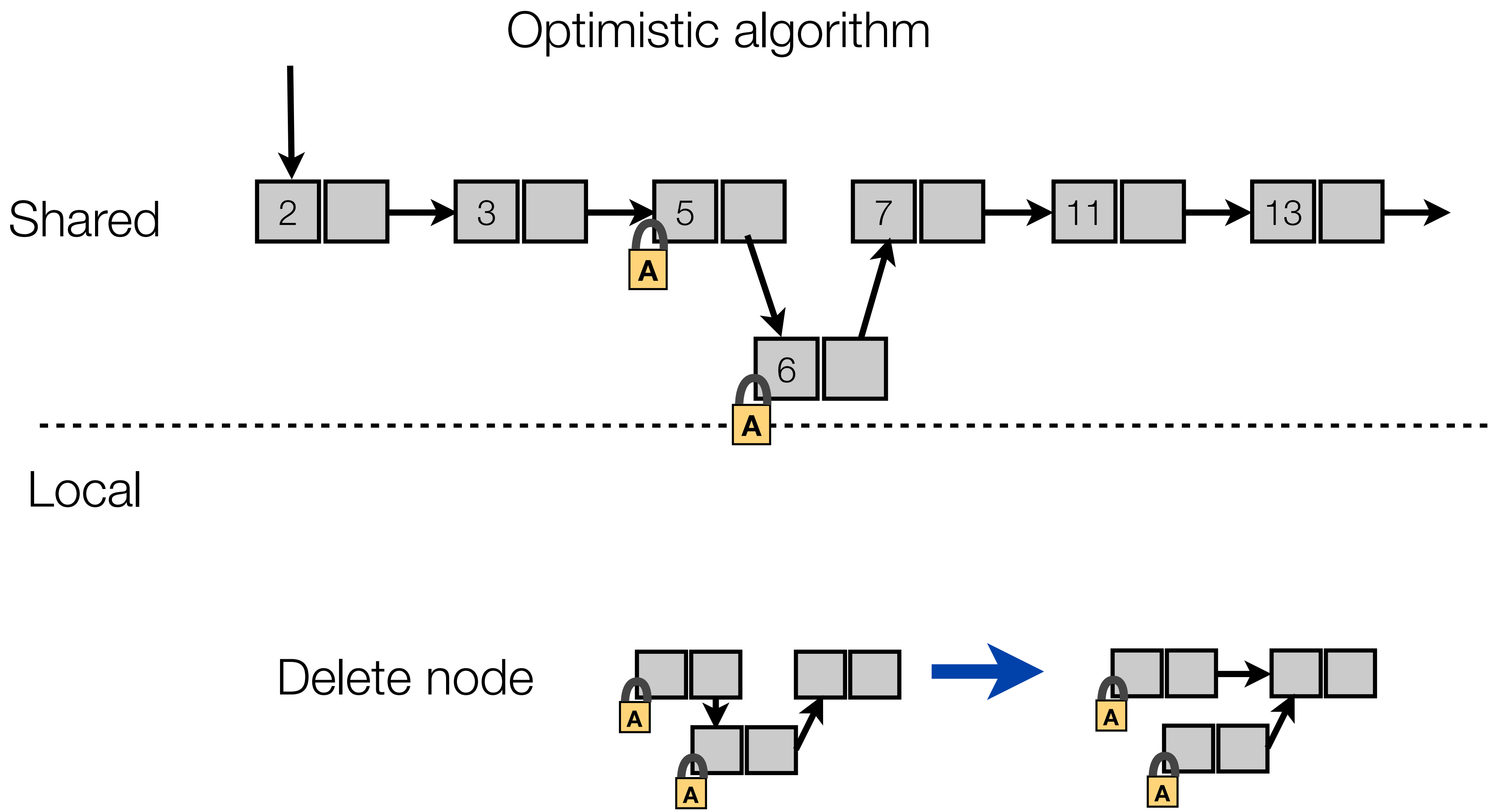
Add node



Delete node

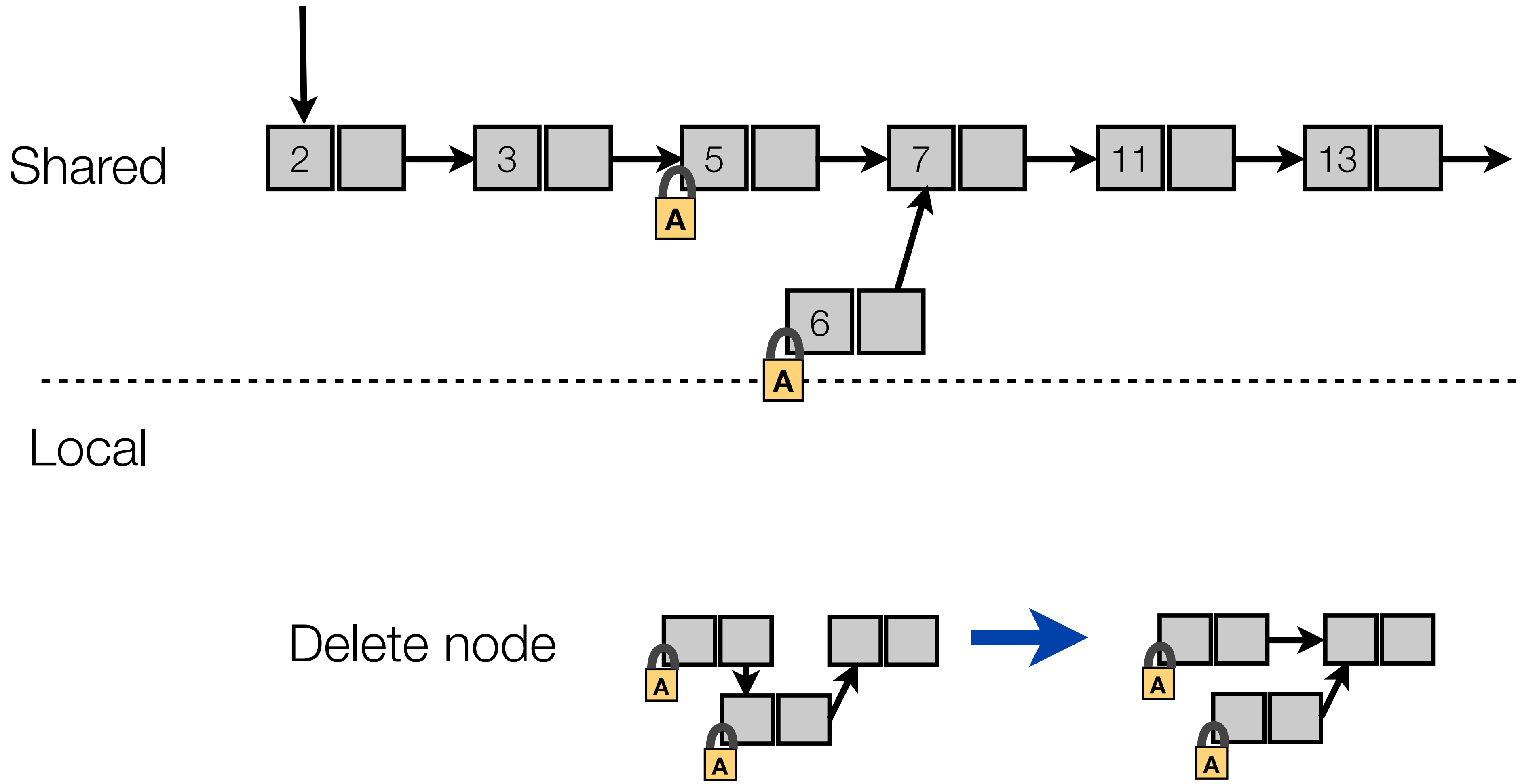


Local and shared state

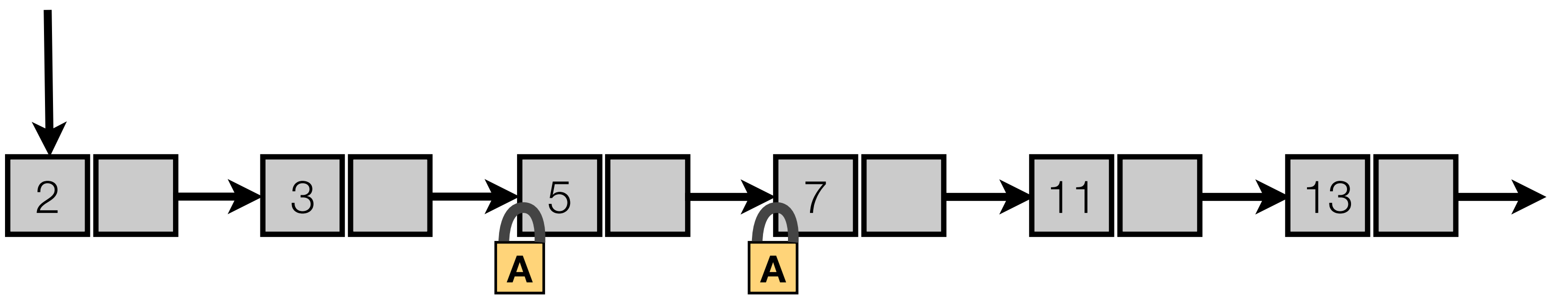


Local and shared state

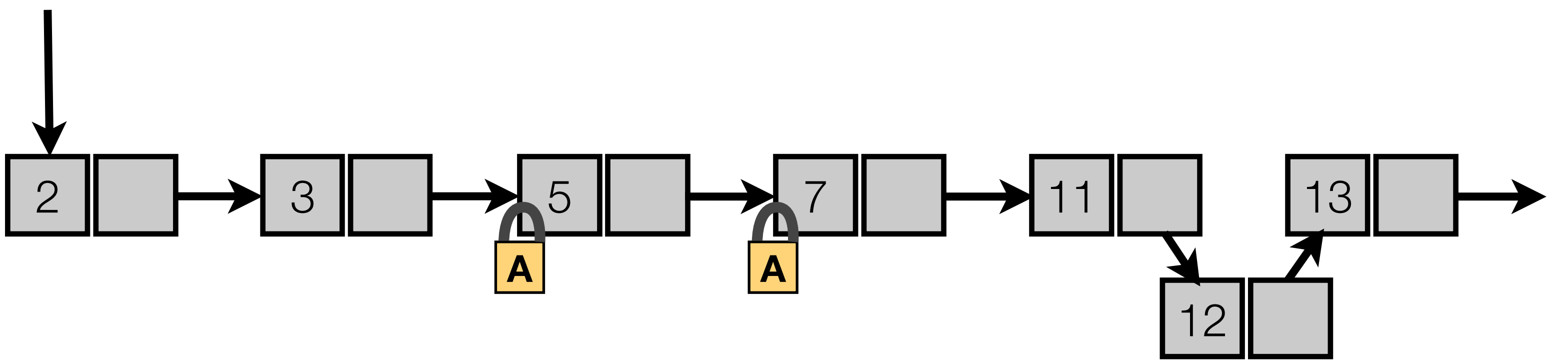
Optimistic algorithm



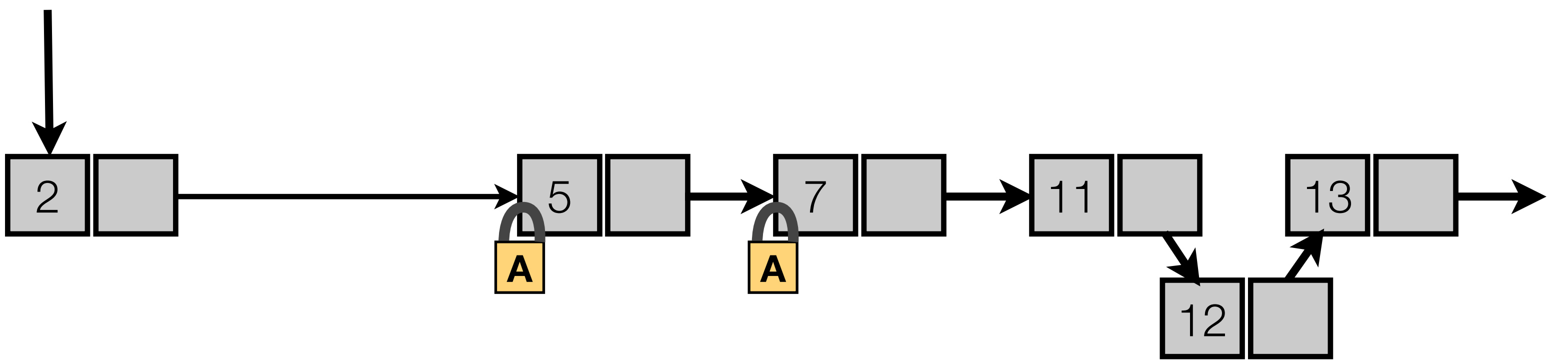
Interference: other threads



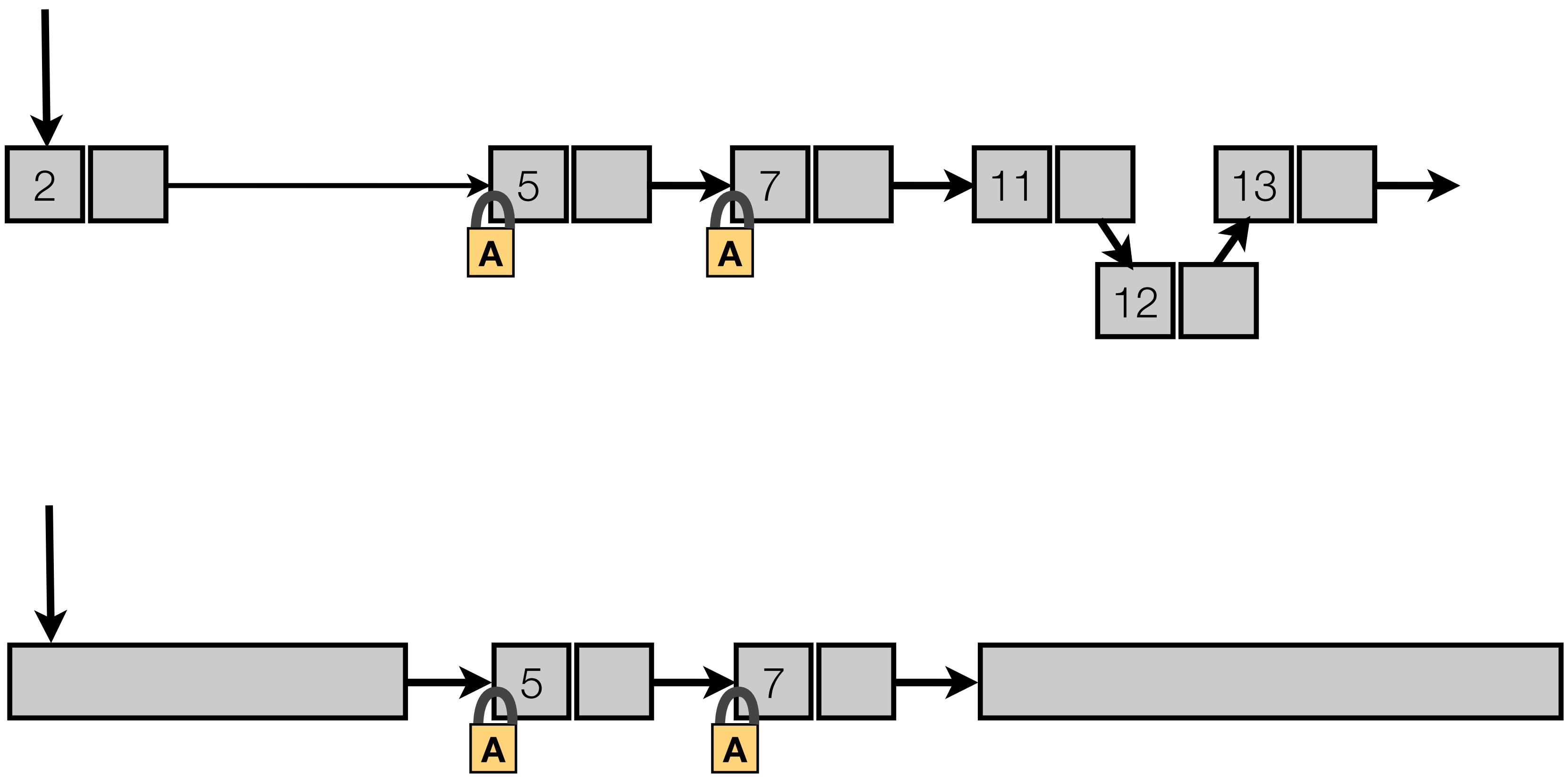
Interference: other threads



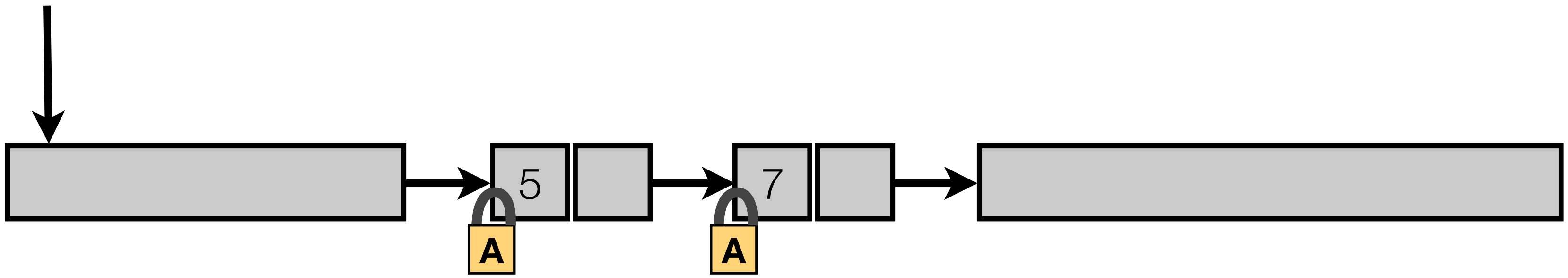
Interference: other threads

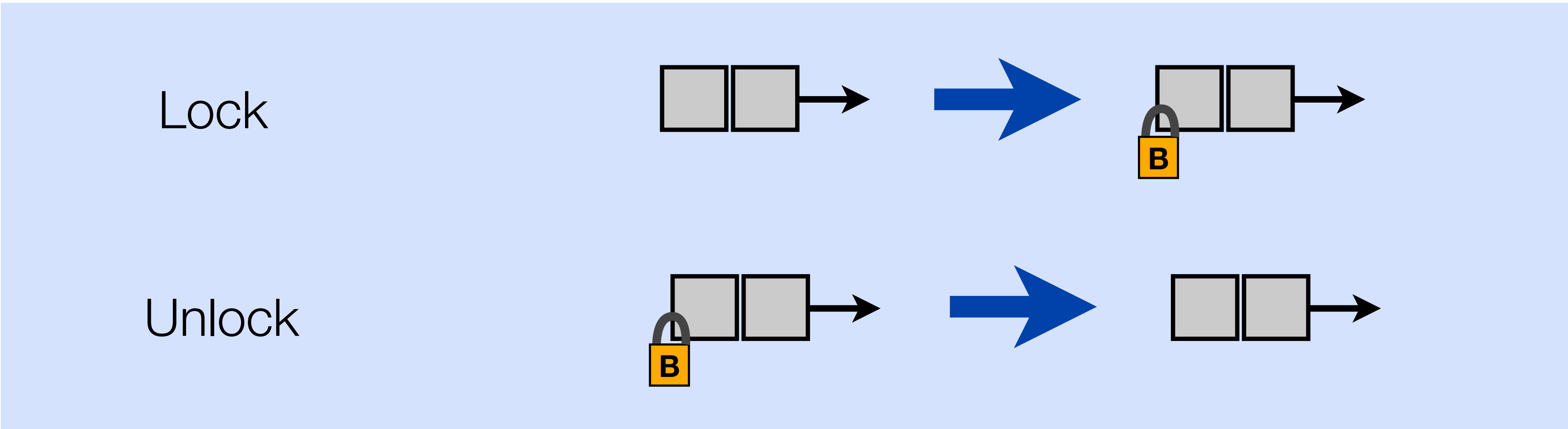


Interference: other threads

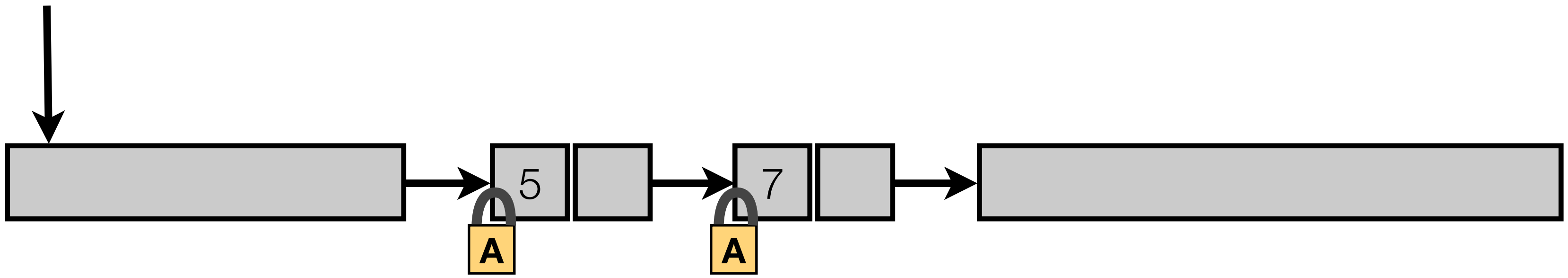


Stability

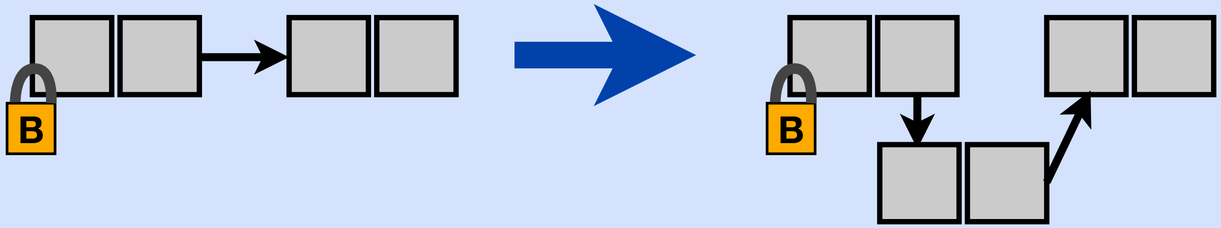




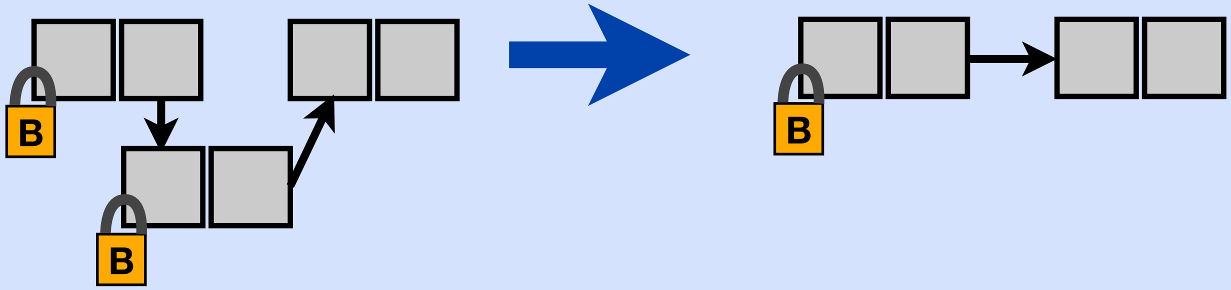
Stability



Add node



Delete node



Assertions

P, Q, R, ... — separation logic assertions
p, q, r, ... — RGSep assertions

shared state assertion

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

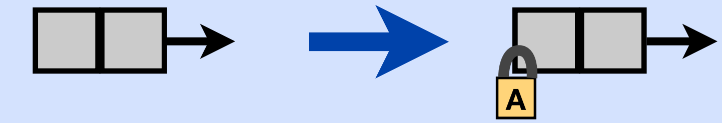
local state assertion

$$P(l, s) \overset{\text{def}}{\Longleftrightarrow} P(l)$$

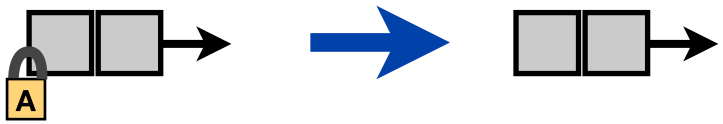
$$\boxed{P}(l, s) \overset{\text{def}}{\Longleftrightarrow} P(s)$$

$$(p * q)(l,s) \overset{\text{def}}{\Longleftrightarrow} \exists l_1\ l_2.\ \mathbf{dom}(l_1) \cap \mathbf{dom}(l_2) = \varnothing \wedge l = l_1 \cup l_2 \wedge p(l_1,s) \wedge q(l_2,s)$$

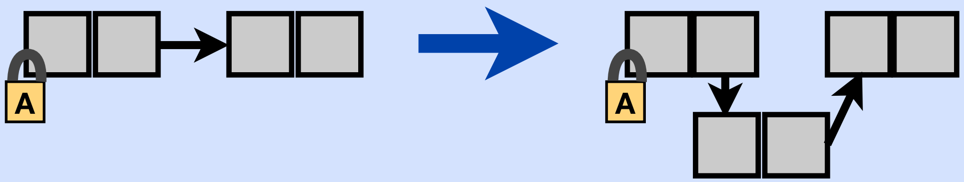
Actions



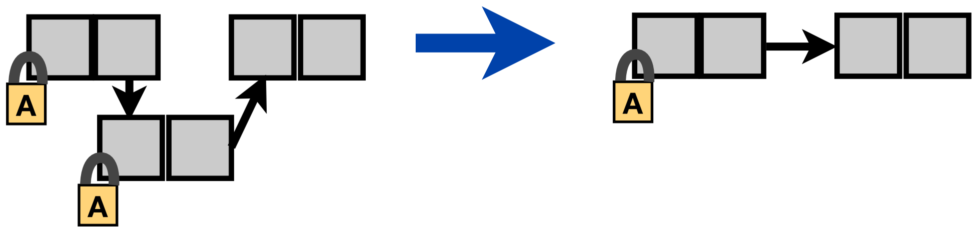
$x \mapsto 0,v,y \quad \rightsquigarrow \quad x \mapsto A,v,y$



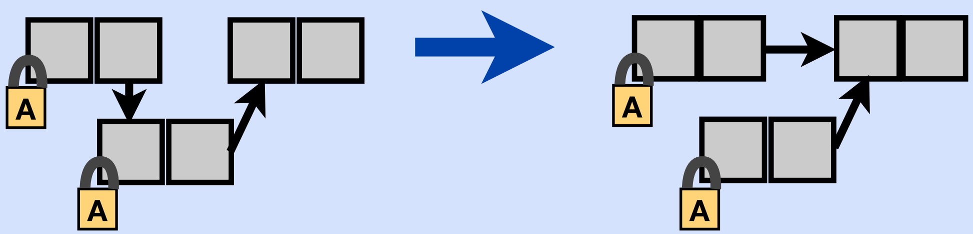
$x \mapsto A,v,y \quad \rightsquigarrow \quad x \mapsto 0,v,y$



$$x \mapsto A,v,y \quad \rightsquigarrow \quad \begin{array}{r} x \mapsto A,v,z \\ * \; z \mapsto 0,w,y \end{array}$$



$$\begin{array}{r} x \mapsto A,v,y \\ * \; y \mapsto A,w,z \end{array} \quad \rightsquigarrow \quad x \mapsto A,v,z$$



$$\begin{array}{r} x \mapsto A,v,y \\ * \; y \mapsto A,w,z \end{array} \quad \rightsquigarrow \quad \begin{array}{r} x \mapsto A,v,z \\ * \; y \mapsto A,w,z \end{array}$$

Parallel composition

$$\frac{\begin{array}{c} C_1 \ \mathsf{sat}\ (p_1,\ R \cup G_2,\ G_1,\ q_1) \\ C_2 \ \mathsf{sat}\ (p_2,\ R \cup G_1,\ G_2,\ q_2) \end{array}}{(C_1 \parallel C_2)\ \mathsf{sat}\ (p_1 * p_2,\ R,\ G_1 \cup G_2,\ q_1 * q_2)}$$

Atomic commands

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

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

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

Stability

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

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

if and only if

$$(P \multimap\!\circledast S) * Q \Rightarrow S$$

$$(P \multimap\!\circledast S)\,(h) \overset{\text{def}}{\Leftrightarrow} \exists h'.\ \mathbf{dom}(h) \cap \mathbf{dom}(h') = \emptyset \wedge P(h') \wedge S(h \cup h')$$

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