

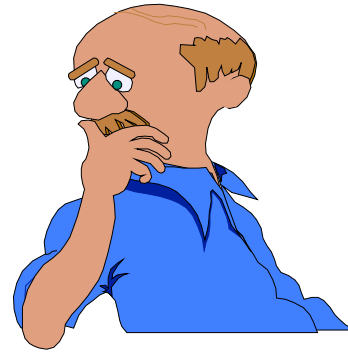
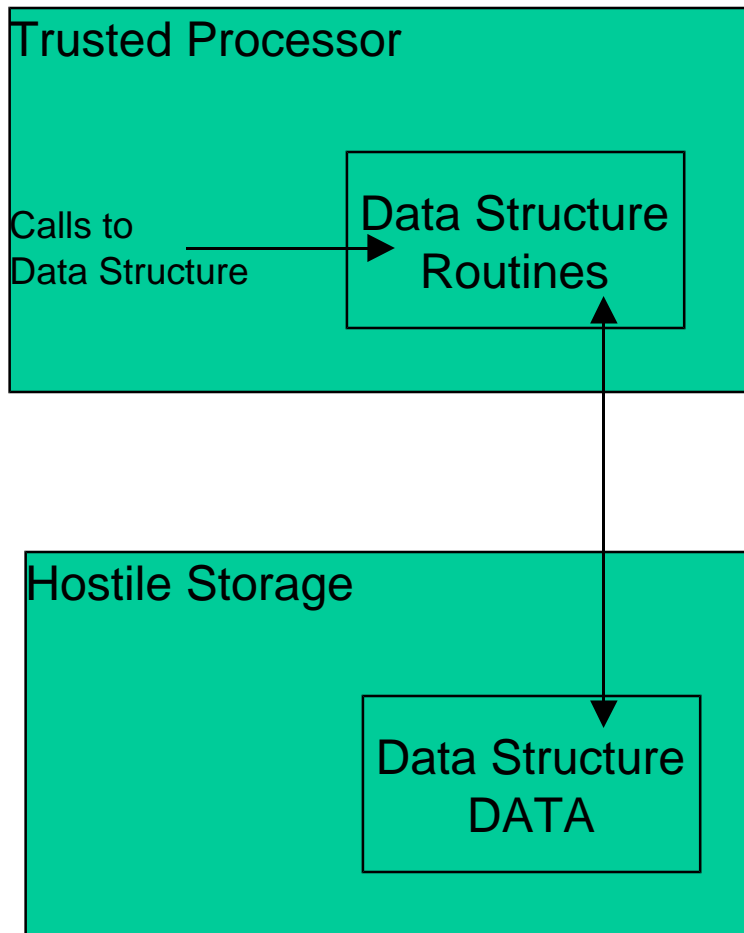
Stack and Queue Integrity on Hostile Platforms

- **Premkumar Devanbu**
- University of California, Davis
- *devanbu@cs.ucdavis.edu*
-
- **Stuart Stubblebine**
- AT&T Laboratories Research.
- *stubblebine@research.att.com*

Outline

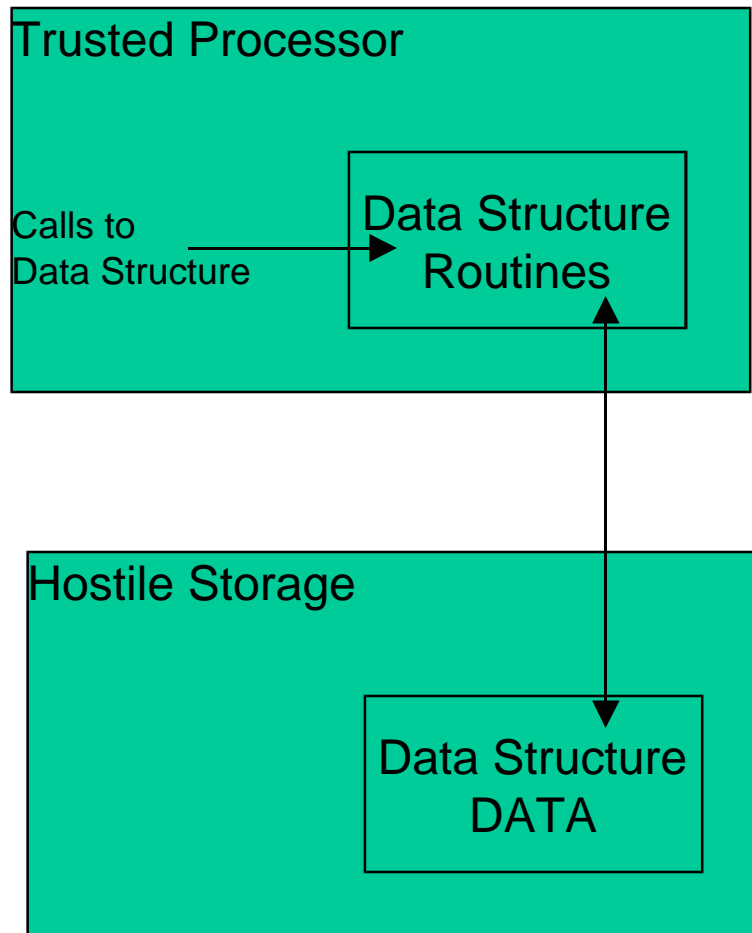
- Problem, threats, motivation.
- Stack method
- Queue method
- Related work
- Conclusions

Problem: Integrity of Data Structures



Problem: Integrity of Data Structures

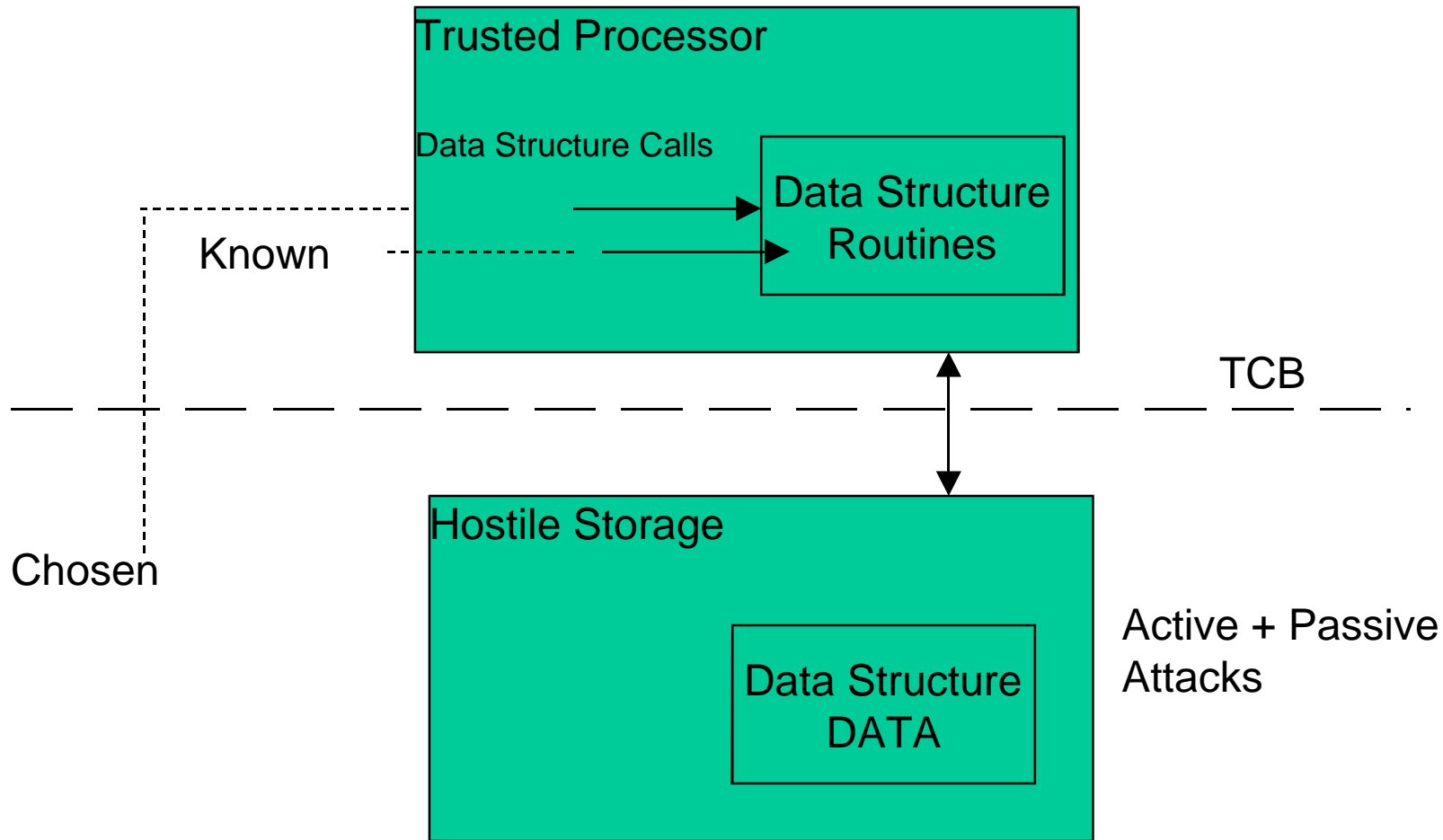
Subject to: Memory + Bandwidth Constraints



Memory Constraints

Bandwidth Constraints

Threats

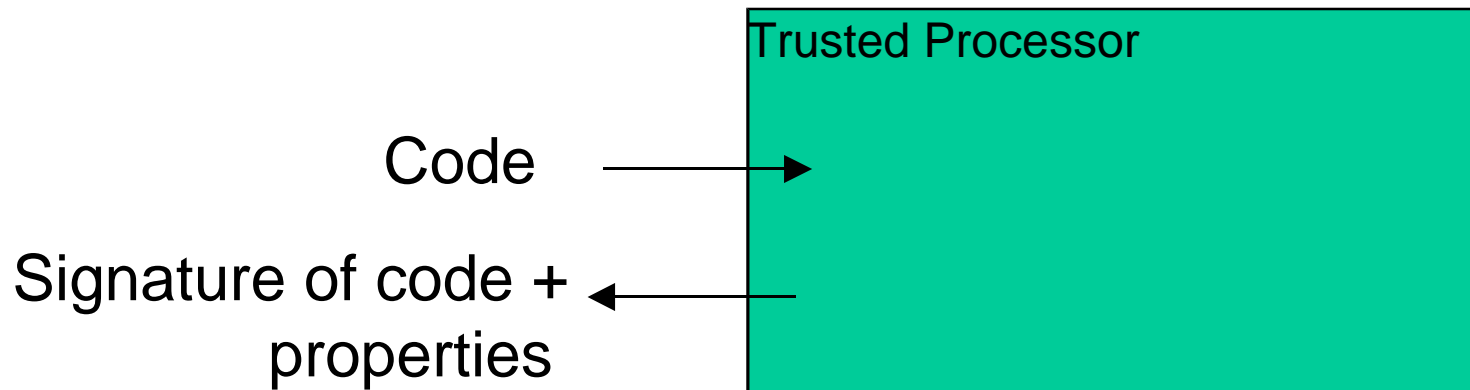


Motivation

- Applications needing to *minimize the expense* of trusted processors (e.g., smart cards, customized chips, etc.) .
 - Memory
 - Bandwidth

Motivating Example: Certifying Properties of Code

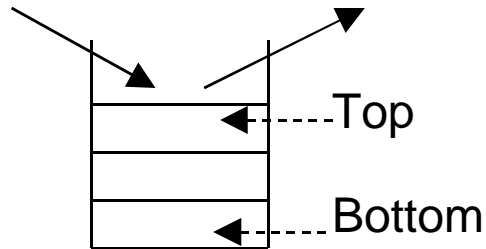
- Trusted processors analyze and certify code provided by software producers.
- Consumers only run code that is certified to have acceptable properties.



Motivating Examples

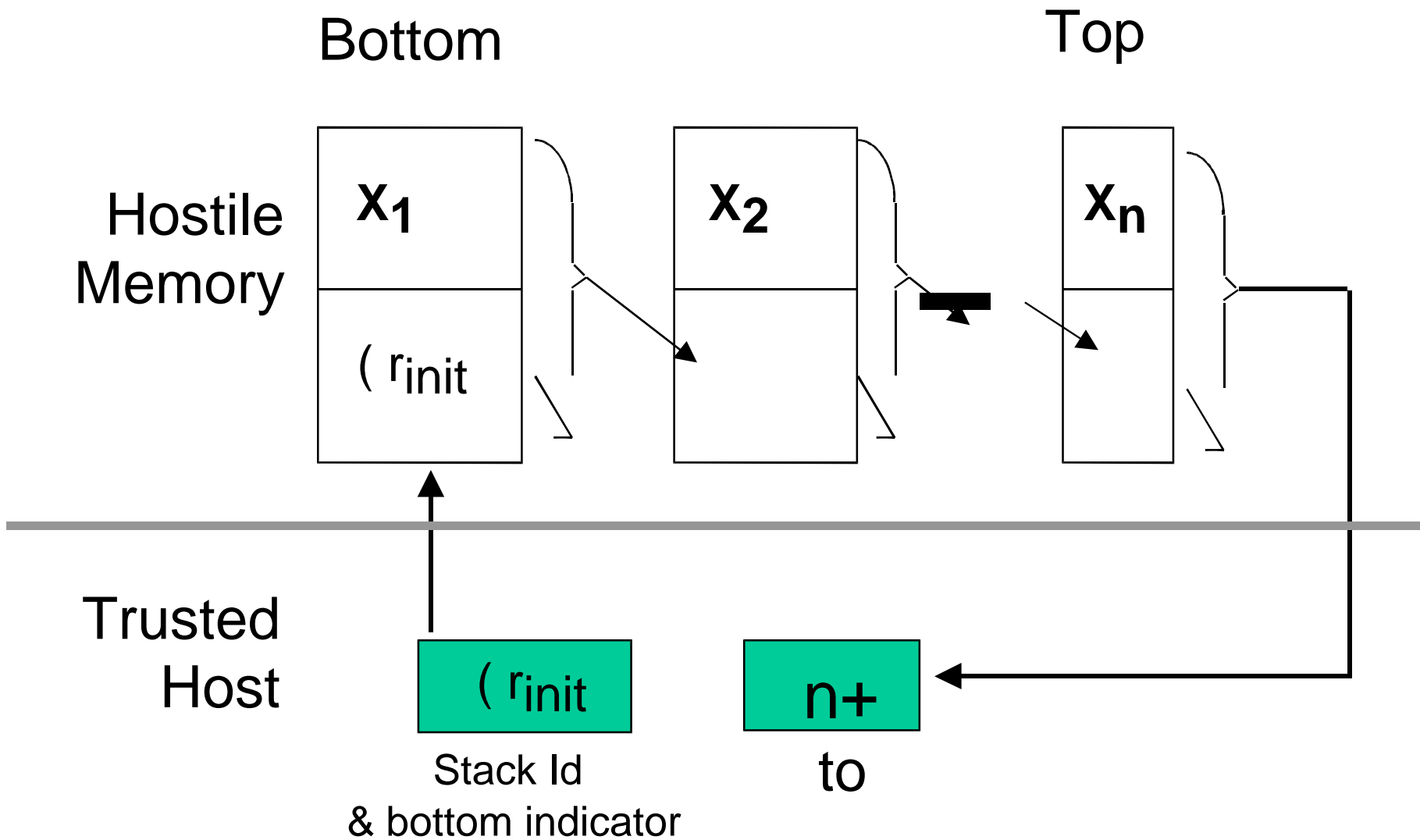
- ^Techniques for trusted software engineering~, *Proceedings of the 20th International Conference on Software Engineering*, 1998.
 - Certifying type checking properties of Java code.
- Applications using:
 - Personal tokens (Java Card, Mondex, etc.)
 - ^Set top box~ hardwar

Stack Description



- Last In First Out (LIFO)
- Operations: push, pop, create, destroy
- Approach
 - maintain (in trusted memory) the signature of the top of the stack
 - upon a pop, verify the top of the stack is correct.
 - we chain elements of the stack together, we include information on the stack that describes the new top of the stack.
 - update the new top of the stack using information returned on a pop.

Overview of Stack Method



Creating a new Stack

T: new()

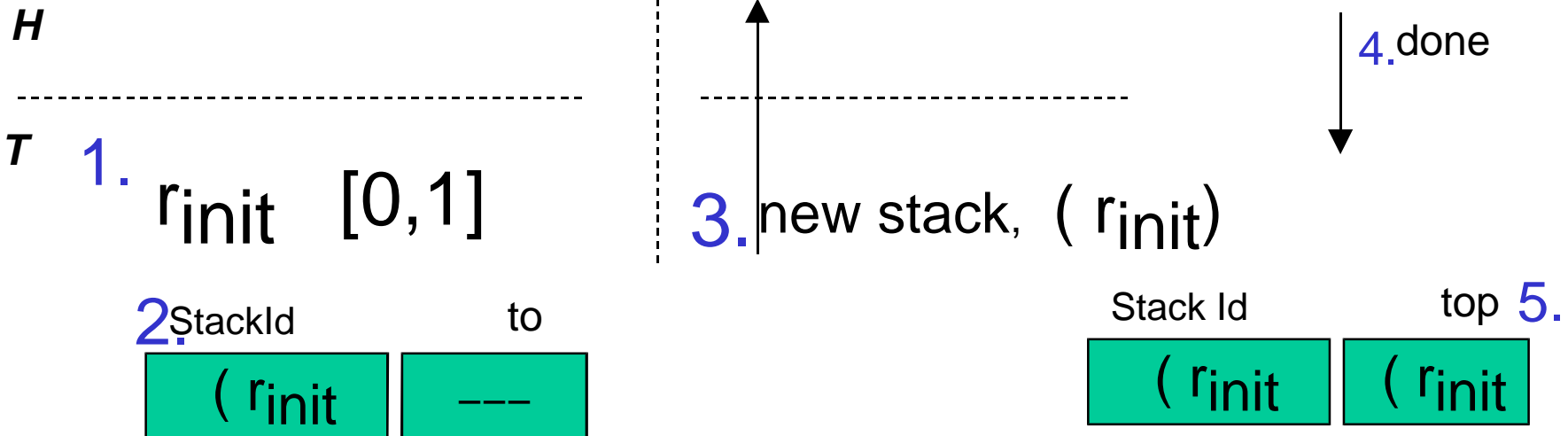
1,2 *T*: $r_{init} [0,1]^l$, StackId (r_{init})

3. *T H*: new stack, StackId

4. *H T*: done

5. *T*: top' (r_{init})

T: StackId



Push

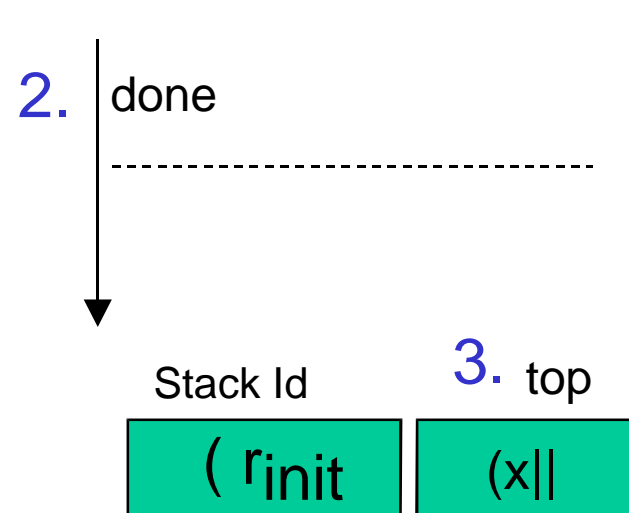
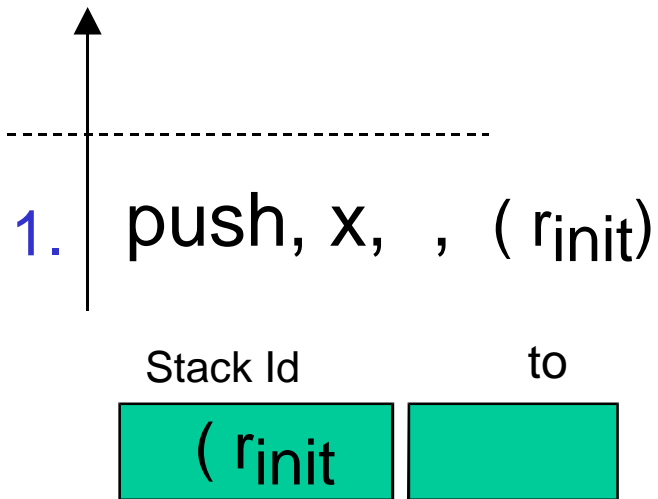
T : push(x , StackId)

1. $T H$: push, x , top , StackId

2. $H T$: done

3. T : $top' = (x || top)$

T : complet



Pop

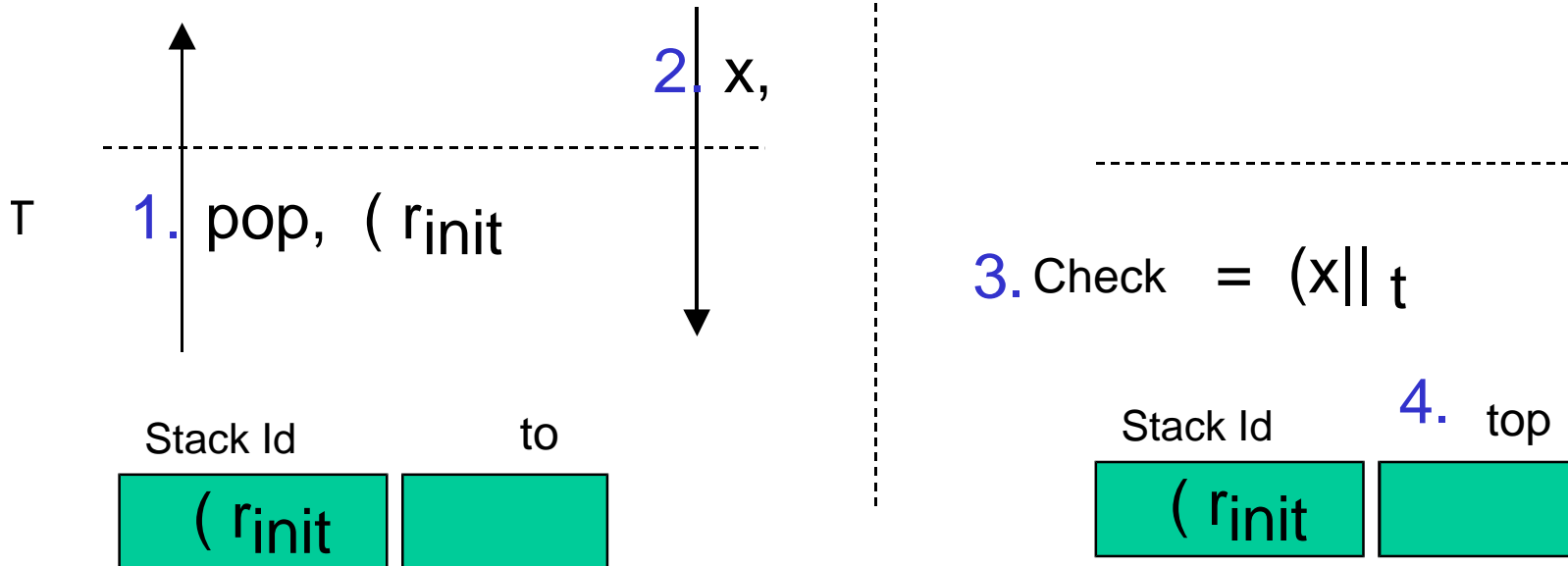
T : pop(StackId)

1. T H : pop, StackId

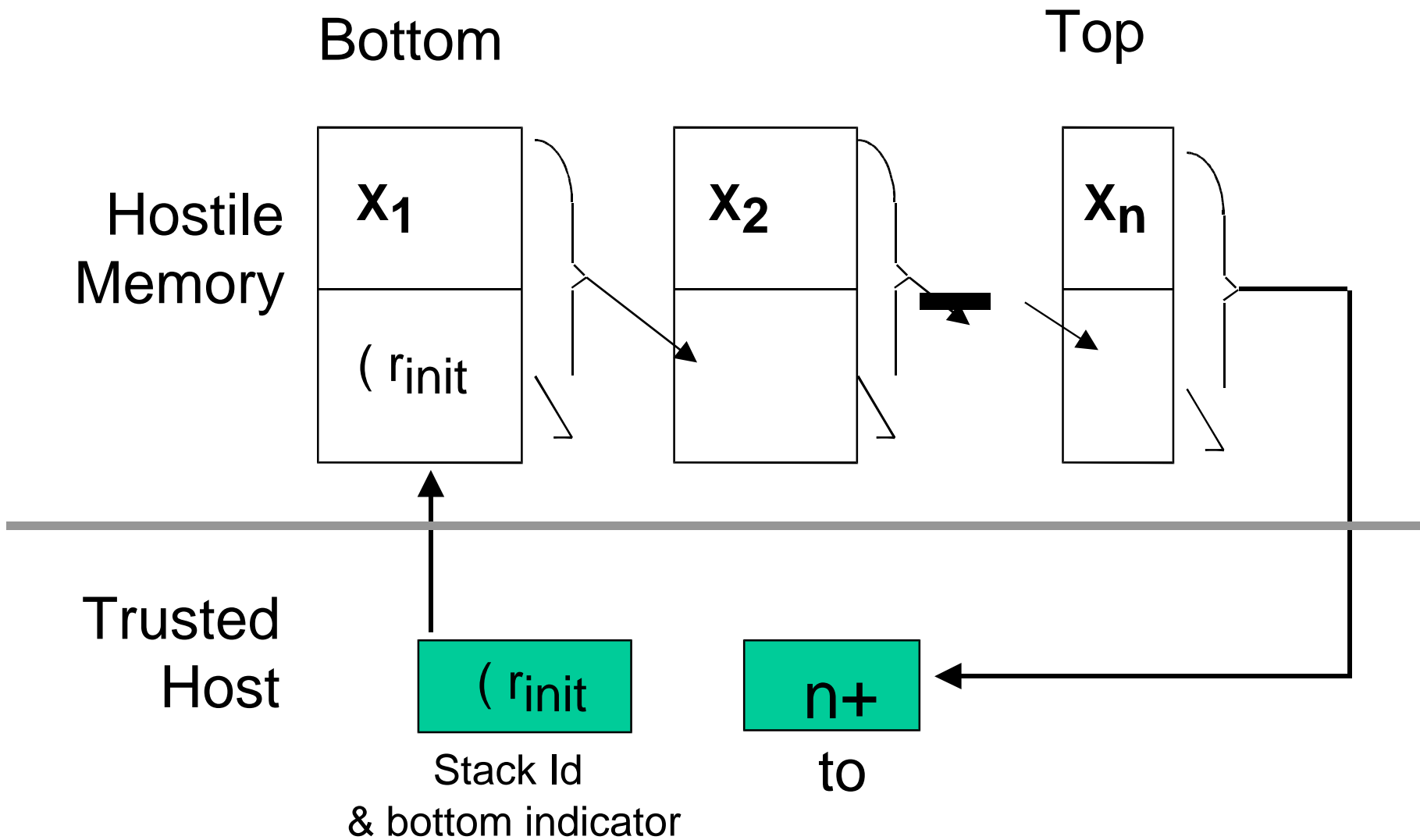
2. H T : x , t

3,4. T : top', t , if top = ($x || t$)

T :



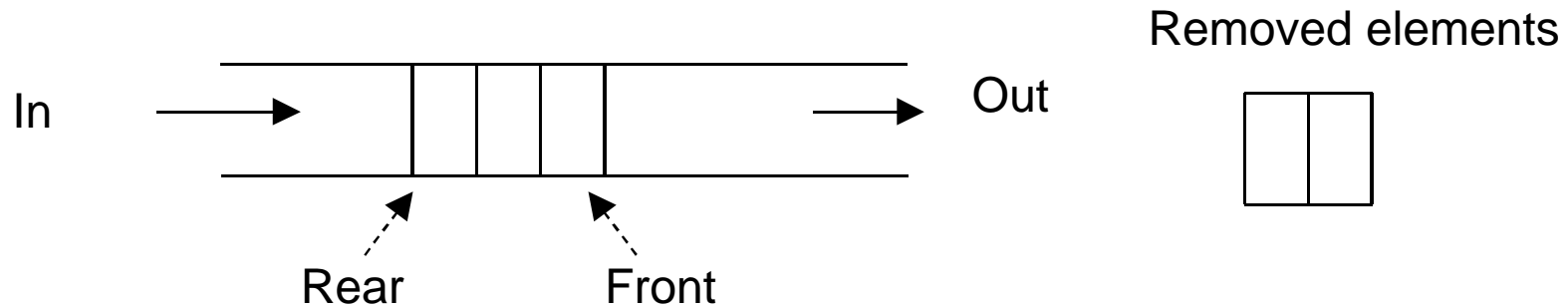
Overview of Stack Method



Evaluation

- Correct if incorrect stack is always detected whenever it returns the wrong value on a pop.
- Operations preserve the stack invariant of maintaining the proper signature for the top of the stack.
- Security due to properties of secure hash/signature functions.
- Replays: same instance or different instance
 - not significant since we know what is the top of the stack.

Queue Description



First In First Out

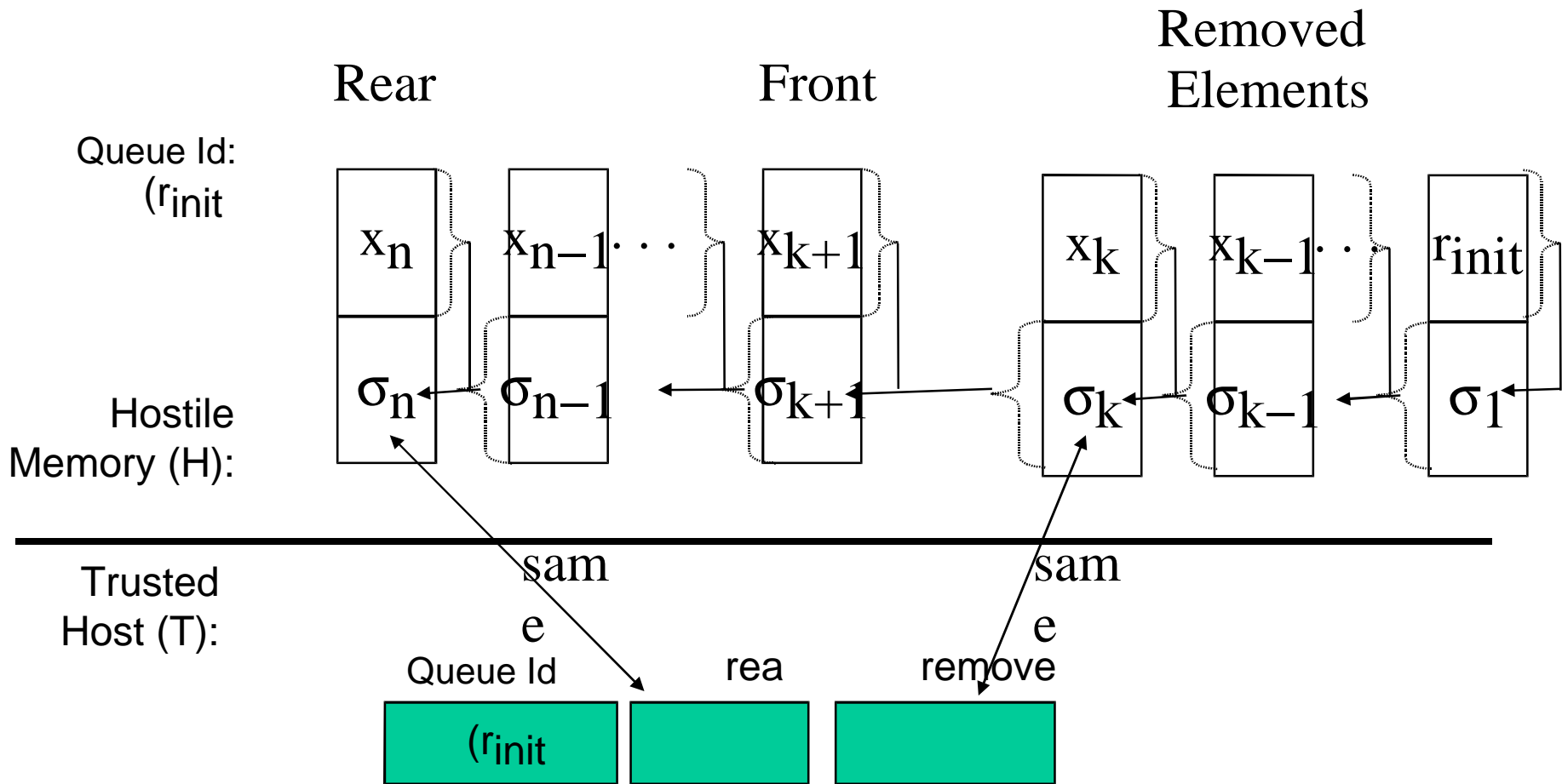
Operations

- enqueue, dequeue, create, destroy

Approach

- Maintain signature of everything added to the queue, **rear**
- Maintain signature of everything removed from the queue, **removed**
- Like the pop operation of a stack, we know what should be next in the queue for a dequeue operation

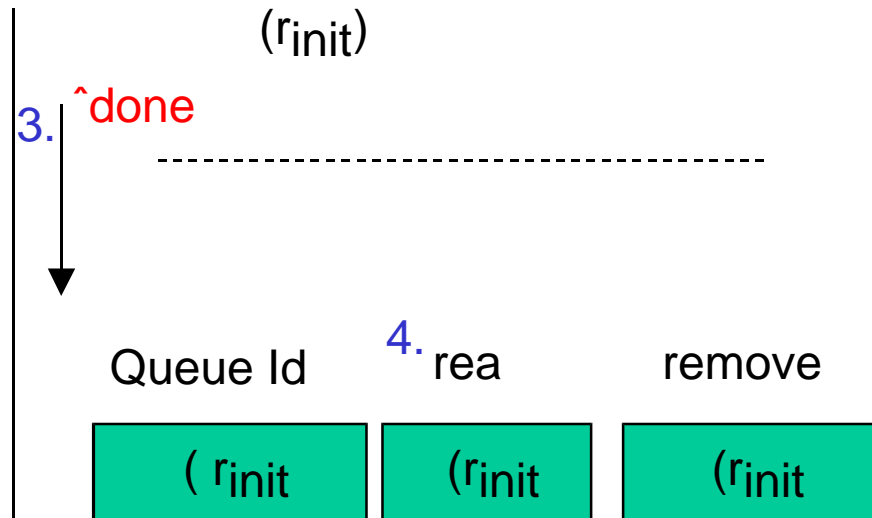
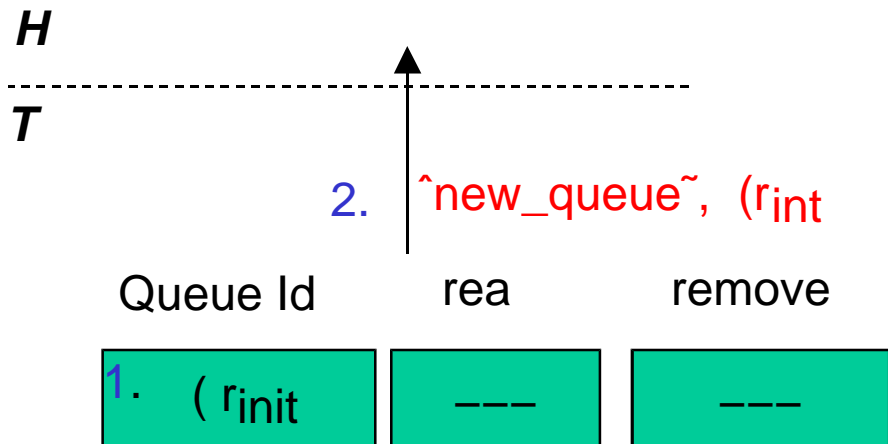
Queue Figure



Create Queue

- T: new_queue()
1. T: rinit R [0,1]^l, QueueId (rinit)
 2. T H: ^new_queue~, QueueId
 3. H T: ^done~
 4. T: rear (rinit), removed (rinit)

T : QueueId
(rinit)



Enqueue

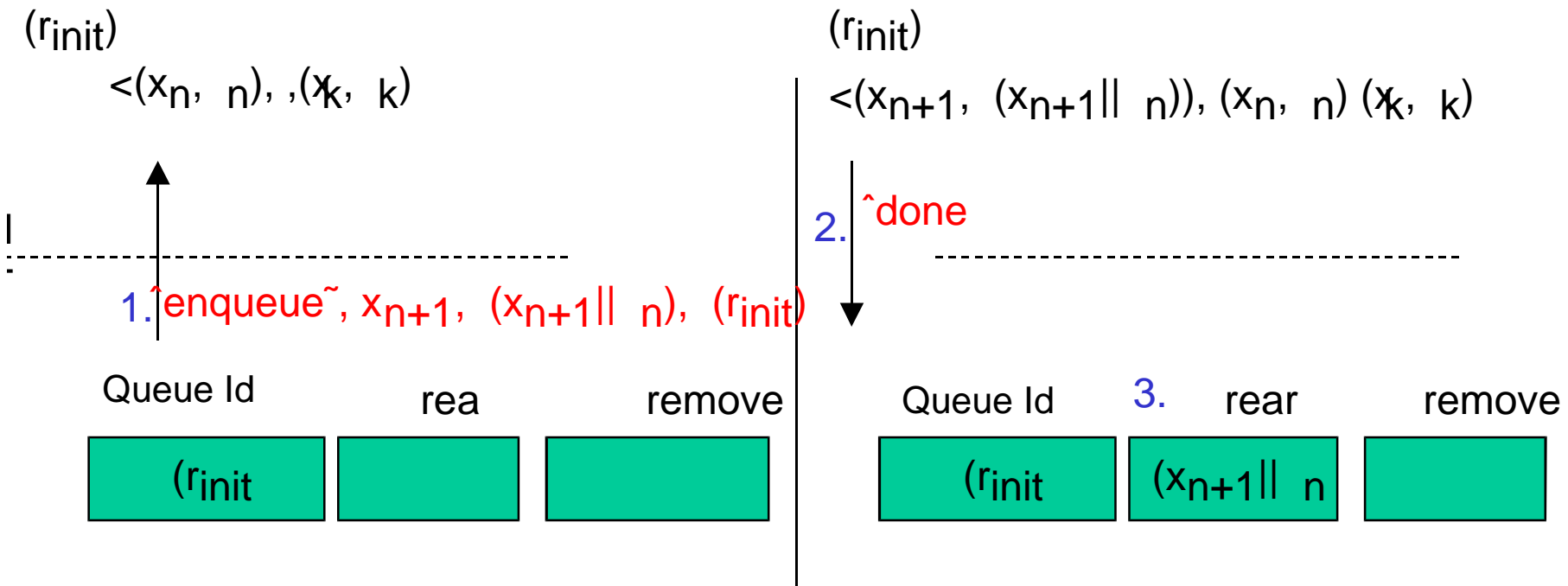
T: enqueue(QueueId,x)

1. T H: $\hat{\text{enqueue}}\sim, x_{n+1}, (x_{n+1} \parallel \text{rear}), \text{QueueId}$

2. H T: $\hat{\text{done}}\sim$

3. T: $\text{rear}' (x_{n+1} \parallel n)$

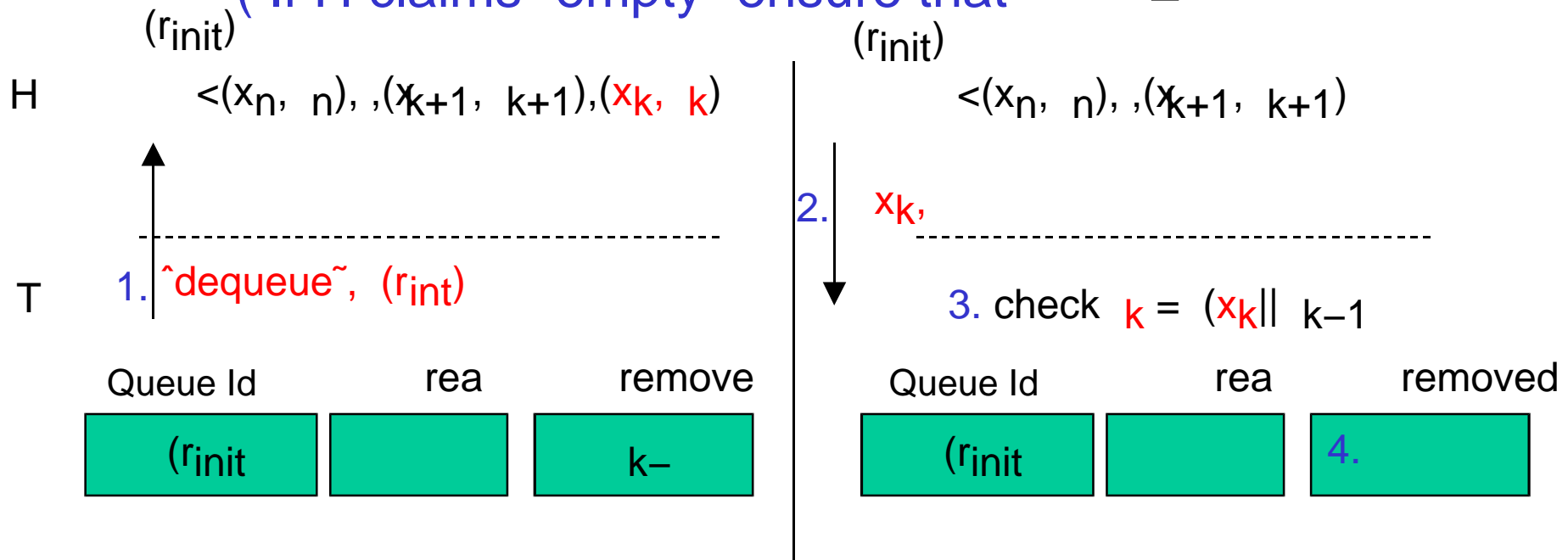
T : complet



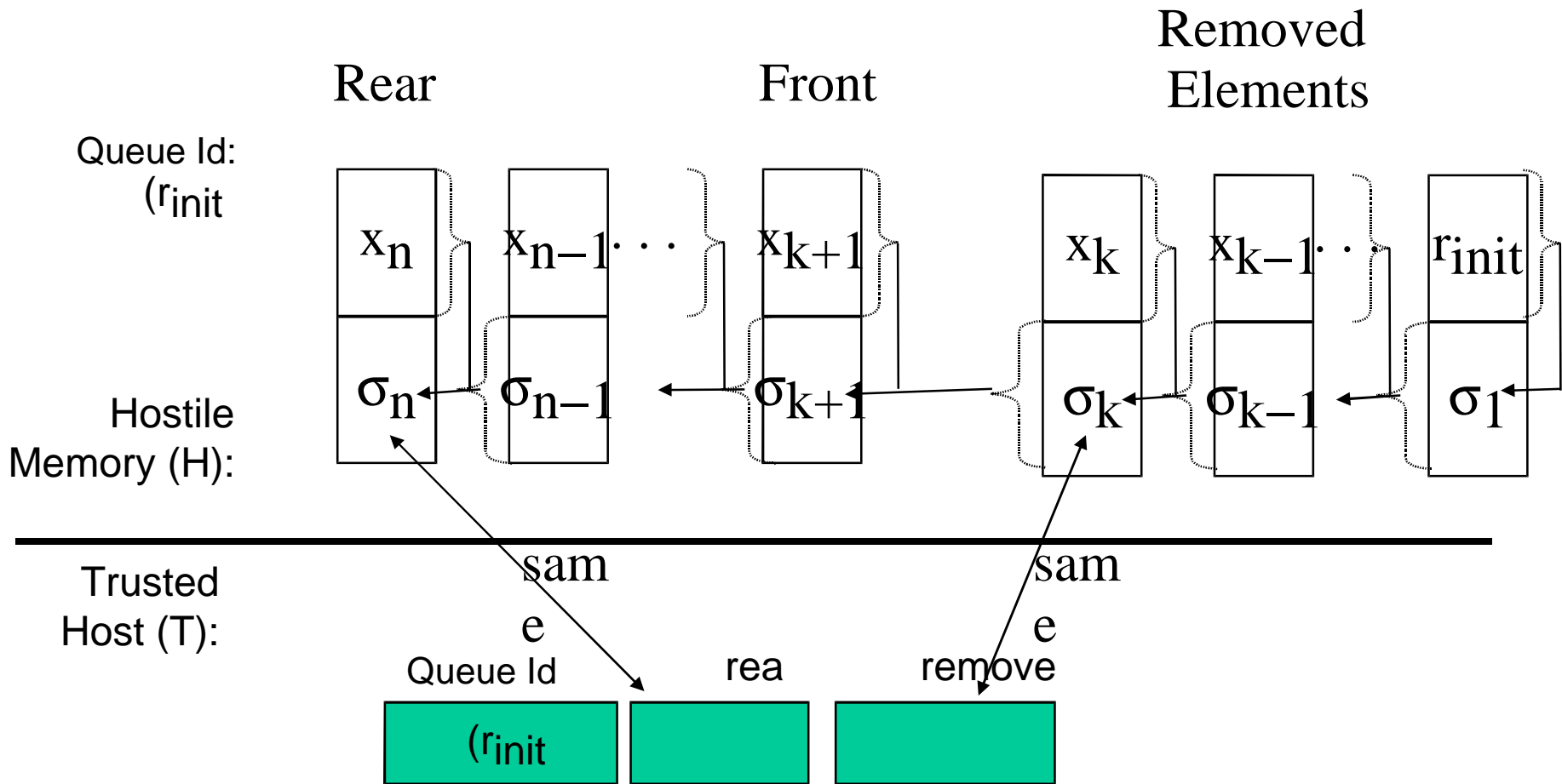
Deque

- T: dequeue(QueueId)
1. T H: $\hat{\text{dequeue}}, \text{QueueId}$
 2. H T: x_k, k
 - 3,4 T: removed k , if $k = (x_k || \text{removed})$
 - T : return x_k

(If H claims $\hat{\text{empty}}$ ensure that $\text{rear} = \text{removed}$



Queue Figure



Related Work

- Checking Memory [Blum et. al. FOCs 91]
 - RAM in $\log n$ based on Merkle hash trees,
 - Queues in $\log n$
- Authentication Chains [Lamport 81]
 - different structure, different application
- Checking Database Integrity [Denning 80s]
 - similar goals, different problem and results

Performance

	Trusted Memory Bits	Transferred Bits	Trusted Operations
Stack	$O(1)$	$O(1)$	$O(1)$
Queue	$O(1)$	$O(1)$	$O(1)$
RAM	$O(\log n)$	$O(\log n)$	$O(\log n)$
Stack using Ram	$O(\log n)$	$O(\log n)$	$O(\log n)$
Queue using Ram	$O(\log n)$	$O(\log n)$	$O(\log n)$

Conclusions

- Methods for stacks and queues of $O(1)$.
- Improvement of $O(\log n)$ over previous approaches.
- Another useful trick to the back of crypto tricks.