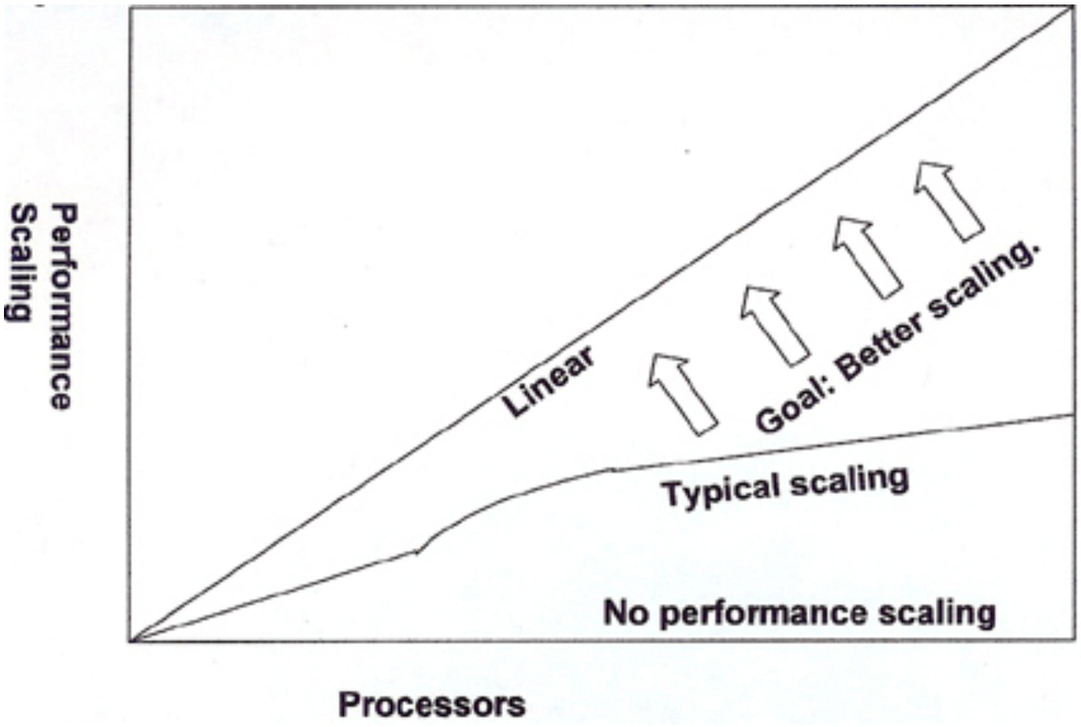
The Adaptive Priority Queue with Elimination and Combining

Irina Calciu, Hammurabi Mendes, Maurice Herlihy
Brown University

Scalability in the Multicore Age

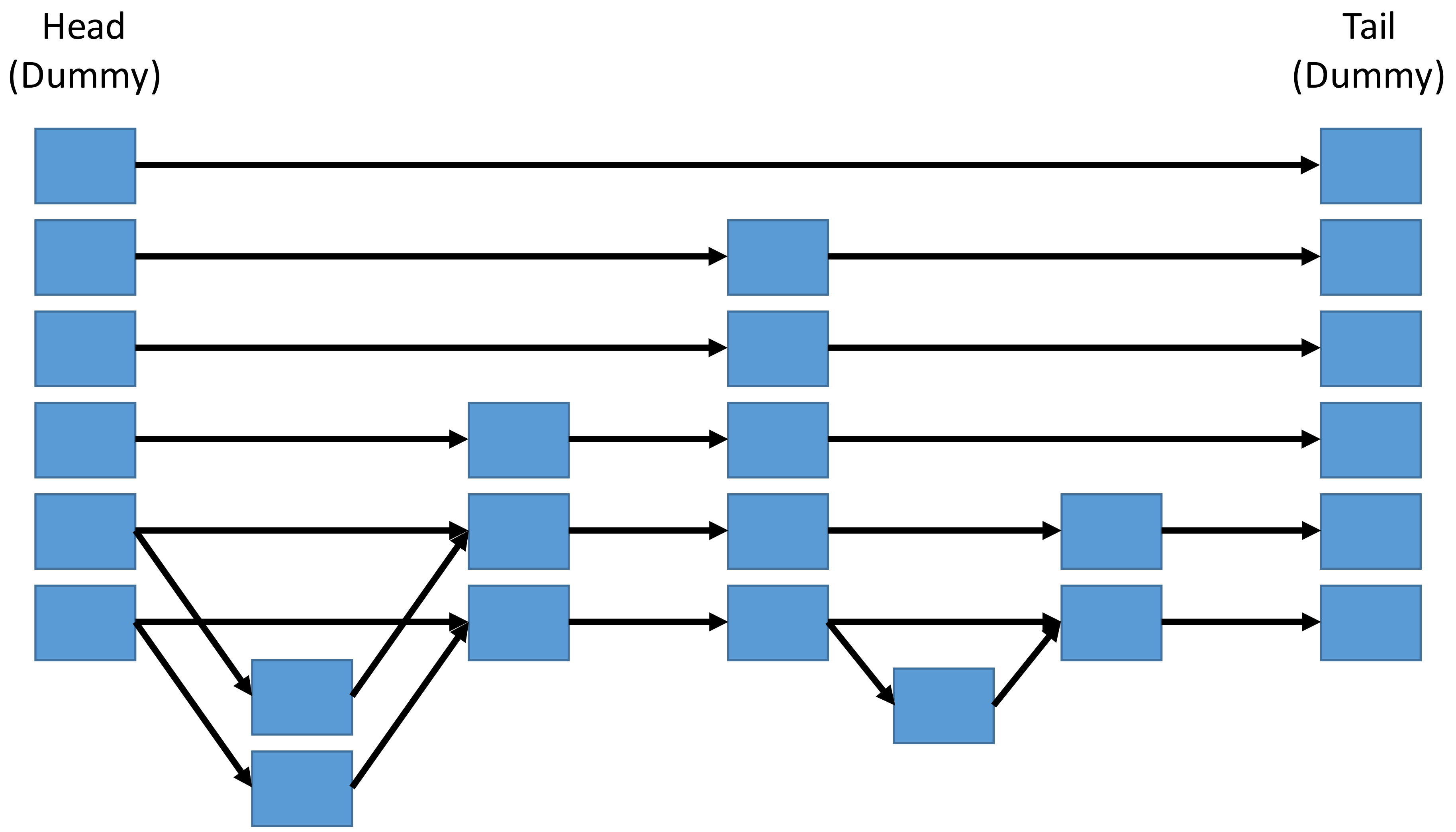


- Our machines are getting bigger, with more cores
- Scalability is far from ideal
- Synchronization is expensive
- We need better data structures for these new architectures

Priority Queue

- Abstract data structure
- Stores <key, value> pairs, where keys are priorities
- Interface: synchronous add(x) and removeMin()
- Implementation: heap or skiplist
- Usage: e.g. resource management

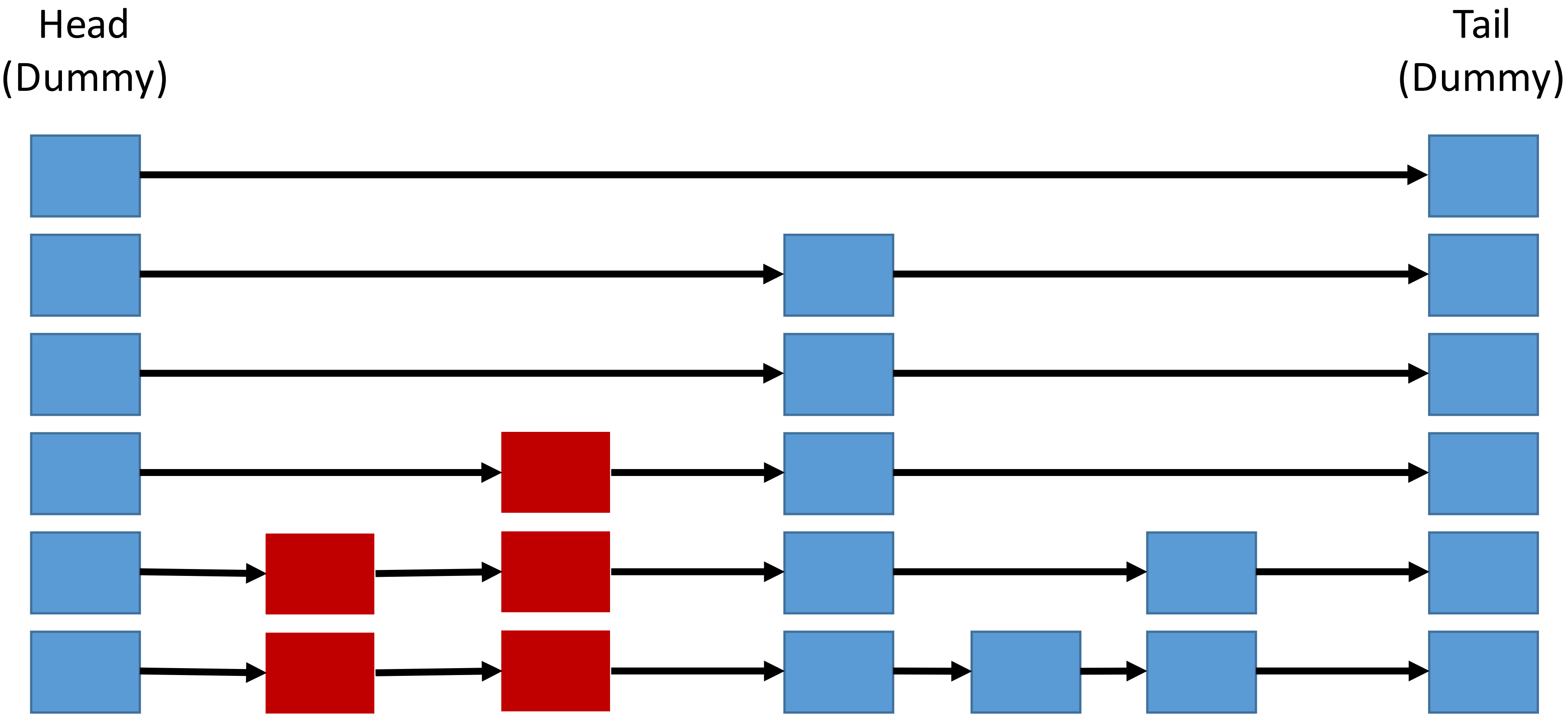
Lazy and Lockfree Skiplists (prior work) – Add()



Efficient parallel add operations

[Lotan2000, Sundell2003]

Lazy and Lockfree Skiplists (prior work) – RemoveMin()

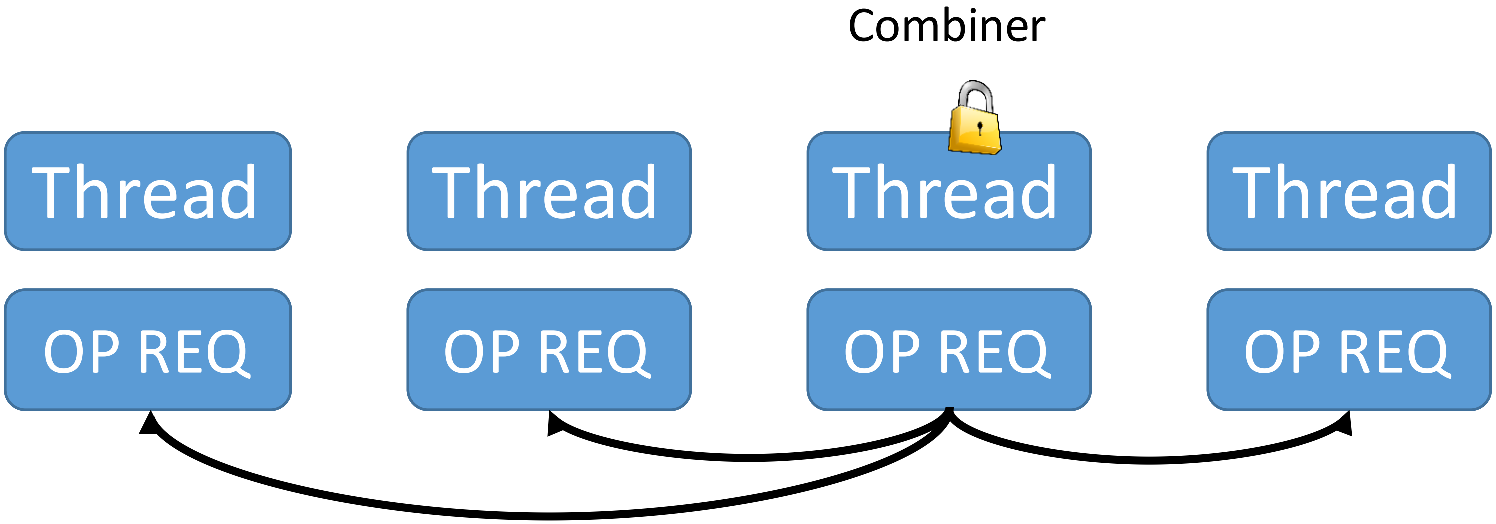

Head
(Dummy)

Tail
(Dummy)

Bottleneck: contention on remove operations

[Lotan2000, Sundell2003]

Flat Combining (prior work)



Combiner

Thread

Thread

Thread

Thread

OP REQ

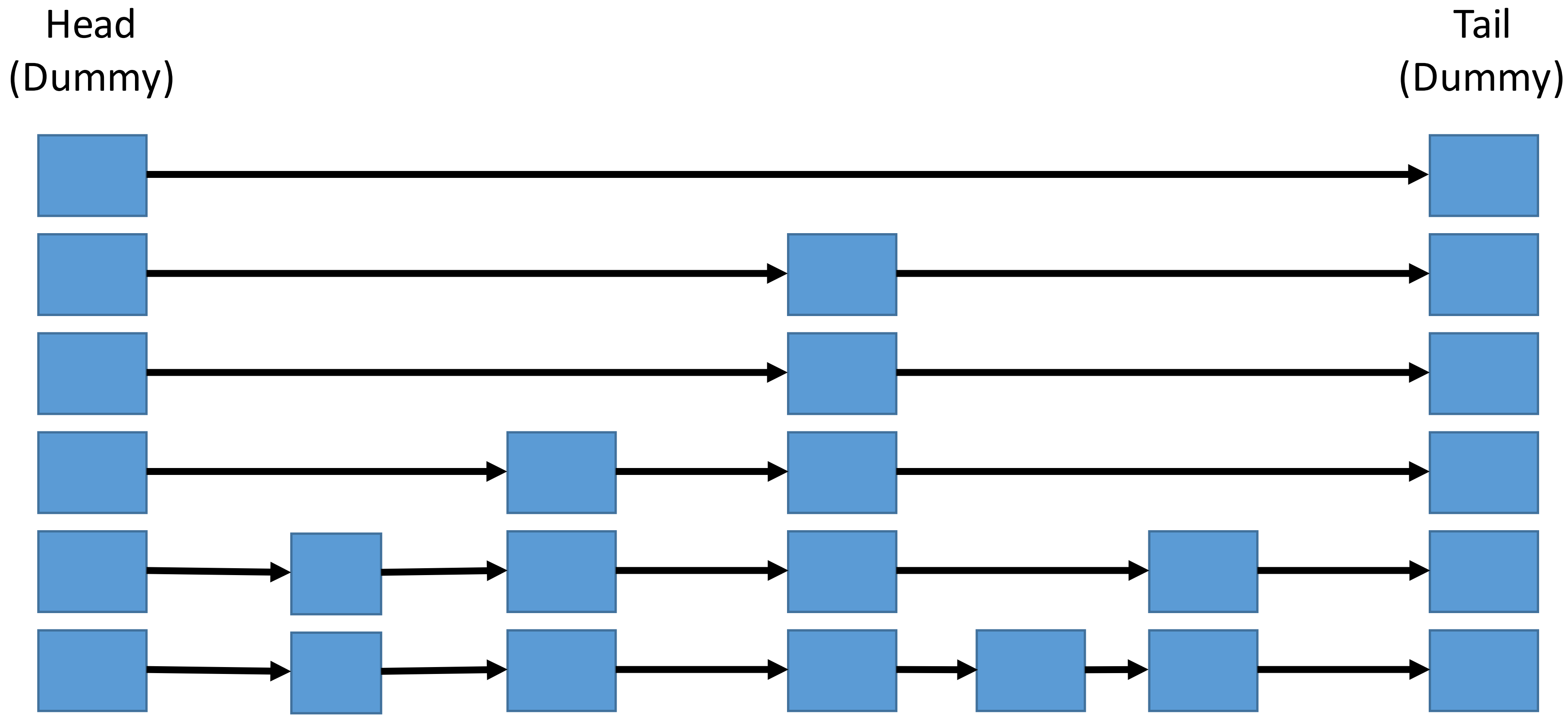
OP REQ

OP REQ

OP REQ

[Hendler2010]

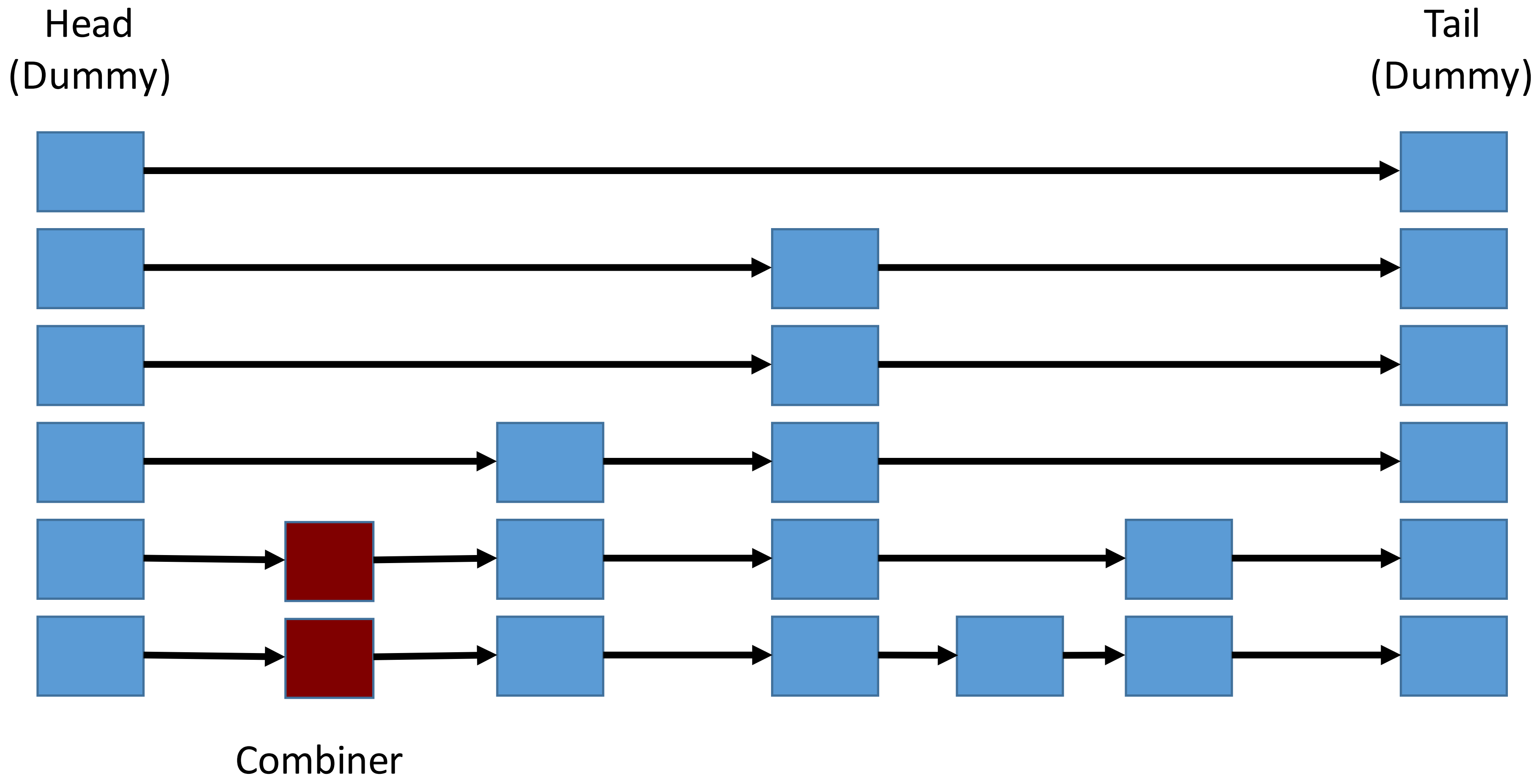
Flat Combining (prior work) – RemoveMin()



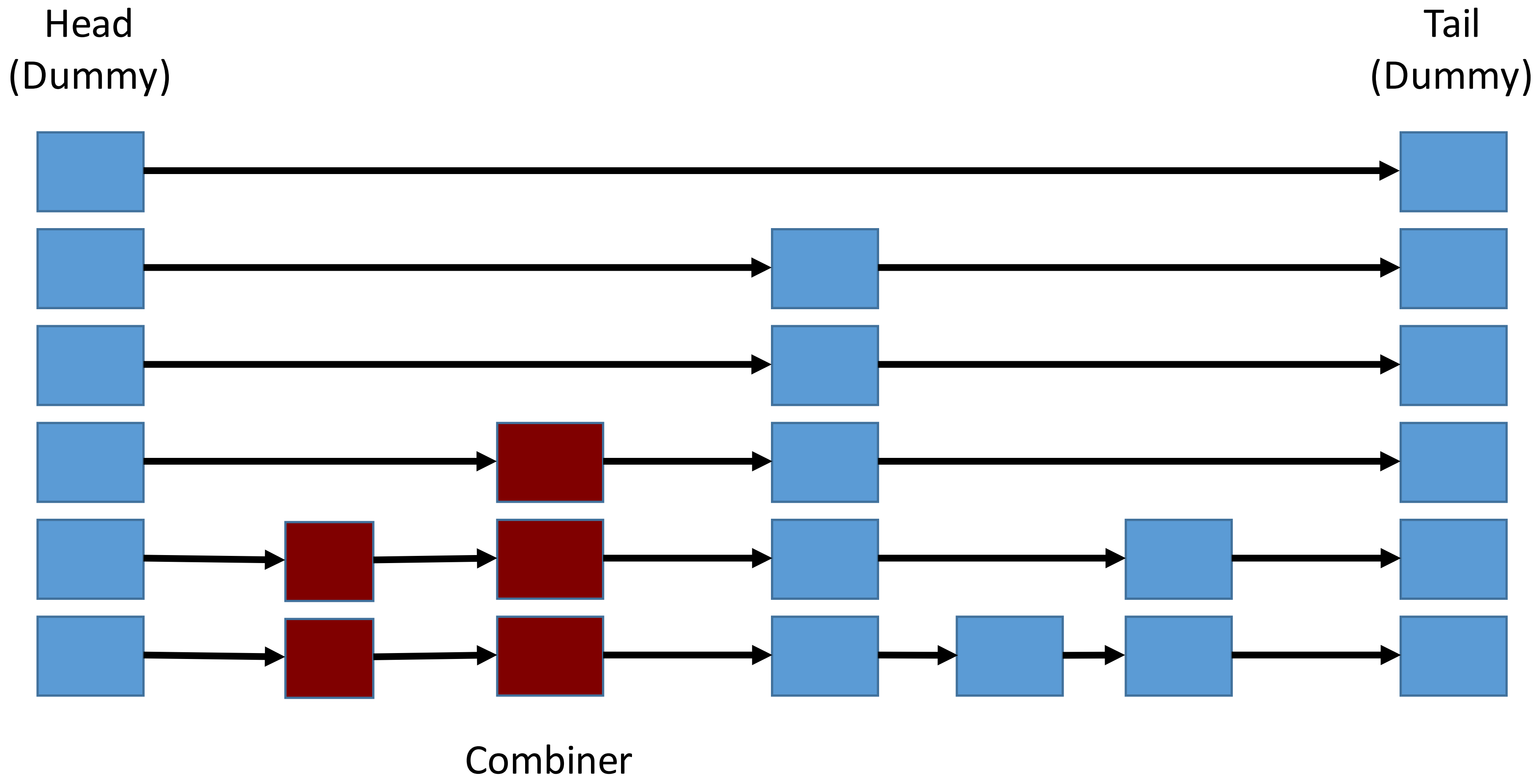
Head
(Dummy)

Tail
(Dummy)

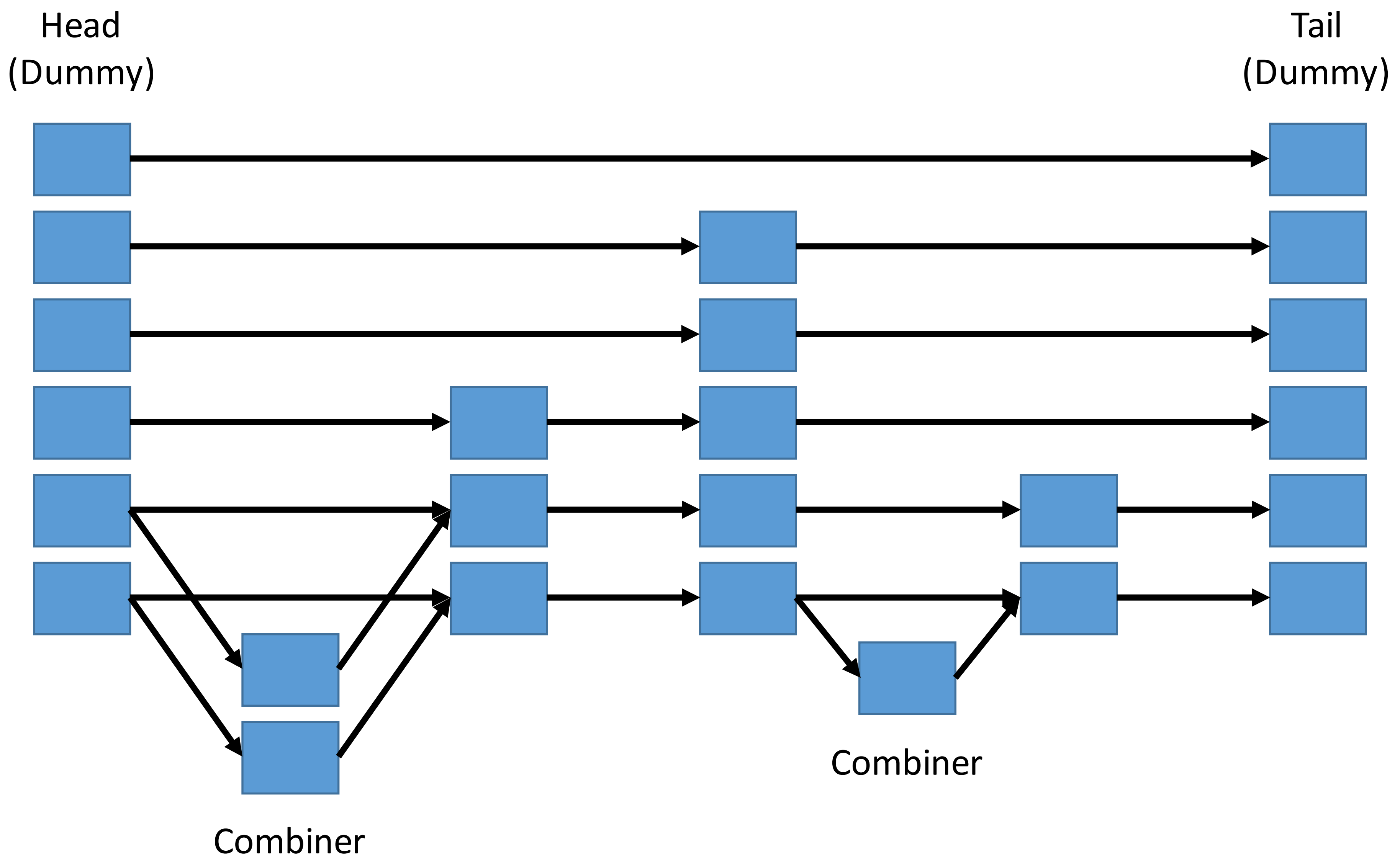
Flat Combining (prior work) – RemoveMin()



Flat Combining (prior work) – RemoveMin()



Flat Combining (prior work) – Add()



Head
(Dummy)

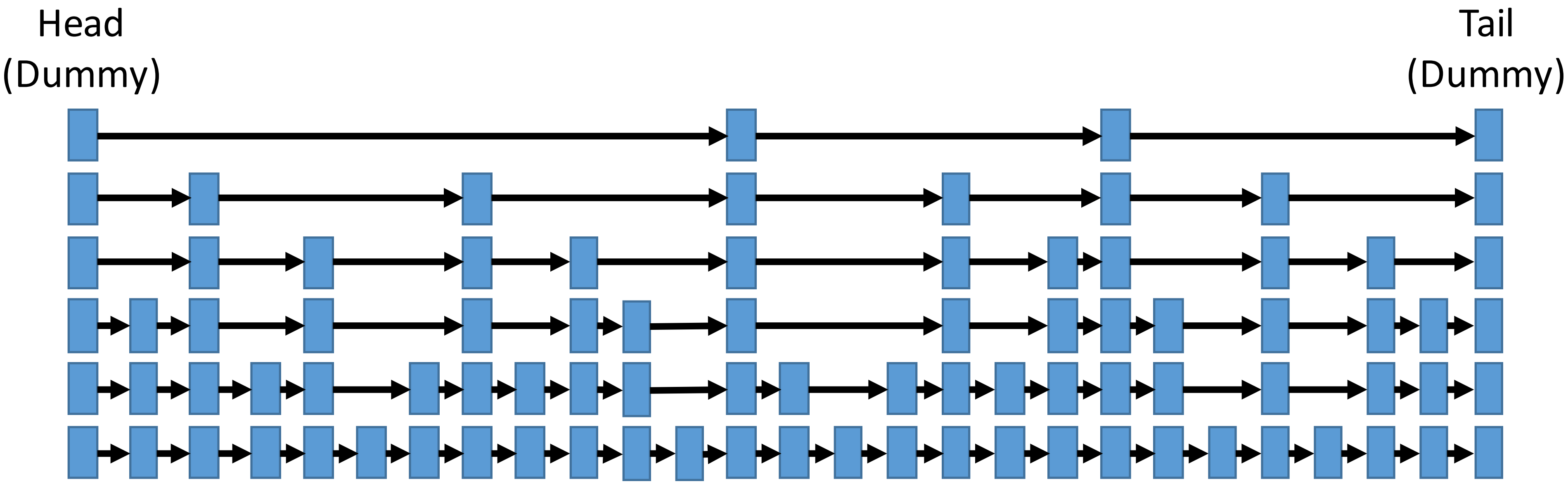
Tail
(Dummy)

Combiner

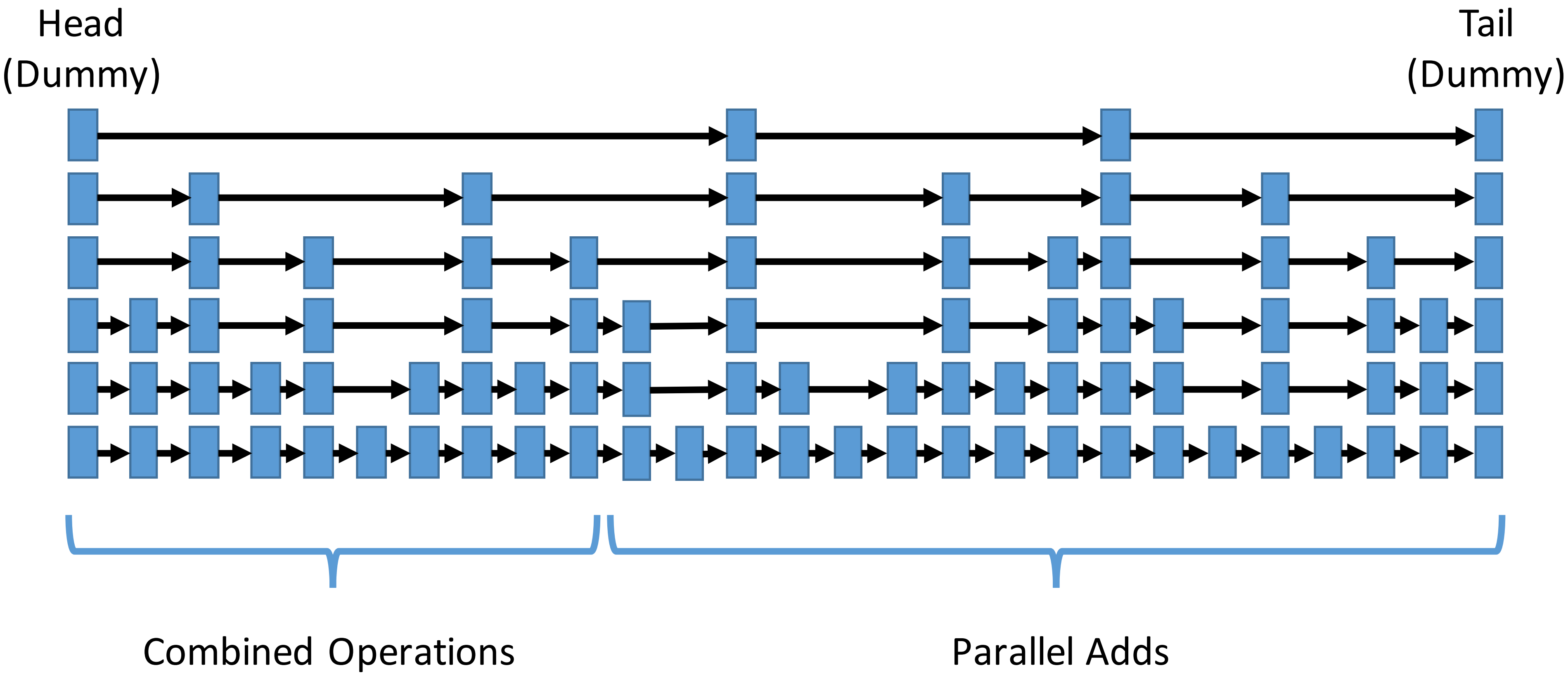
Combiner

Add operations are sequential

Flat Combining (prior work) – Add()



Goal (1): Combining + Parallel Adds

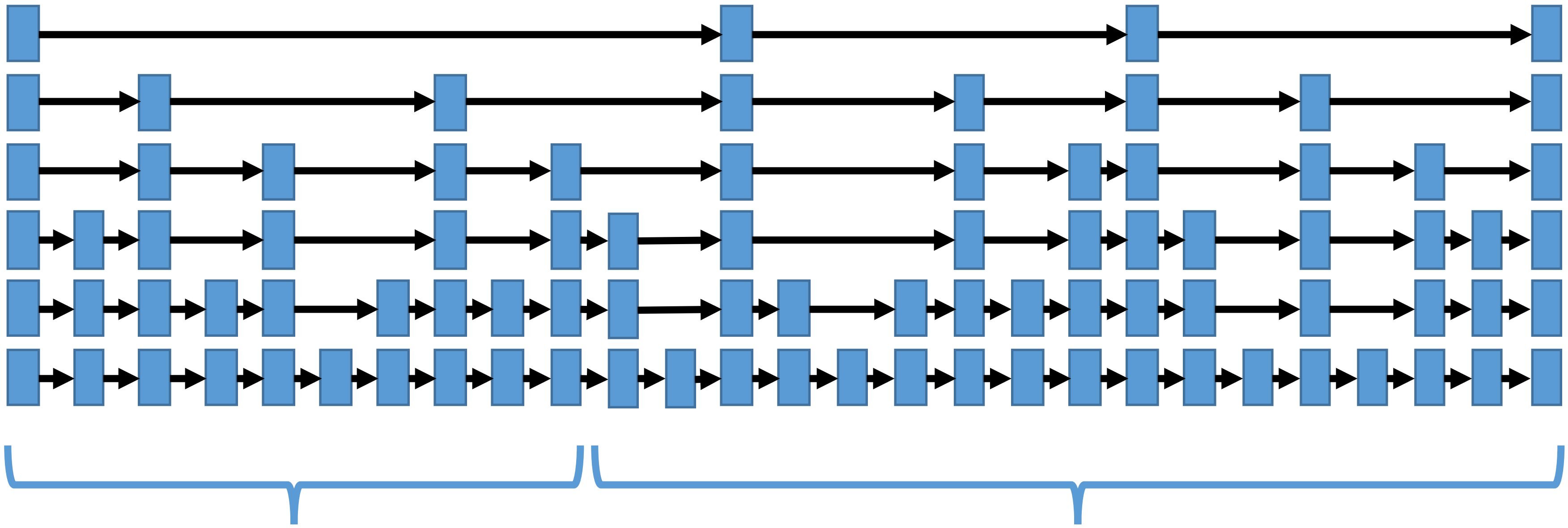

Head
(Dummy)

Tail
(Dummy)

Combined Operations

Parallel Adds

Goal (2): Parallelize Combined Adds Too



Head
(Dummy)

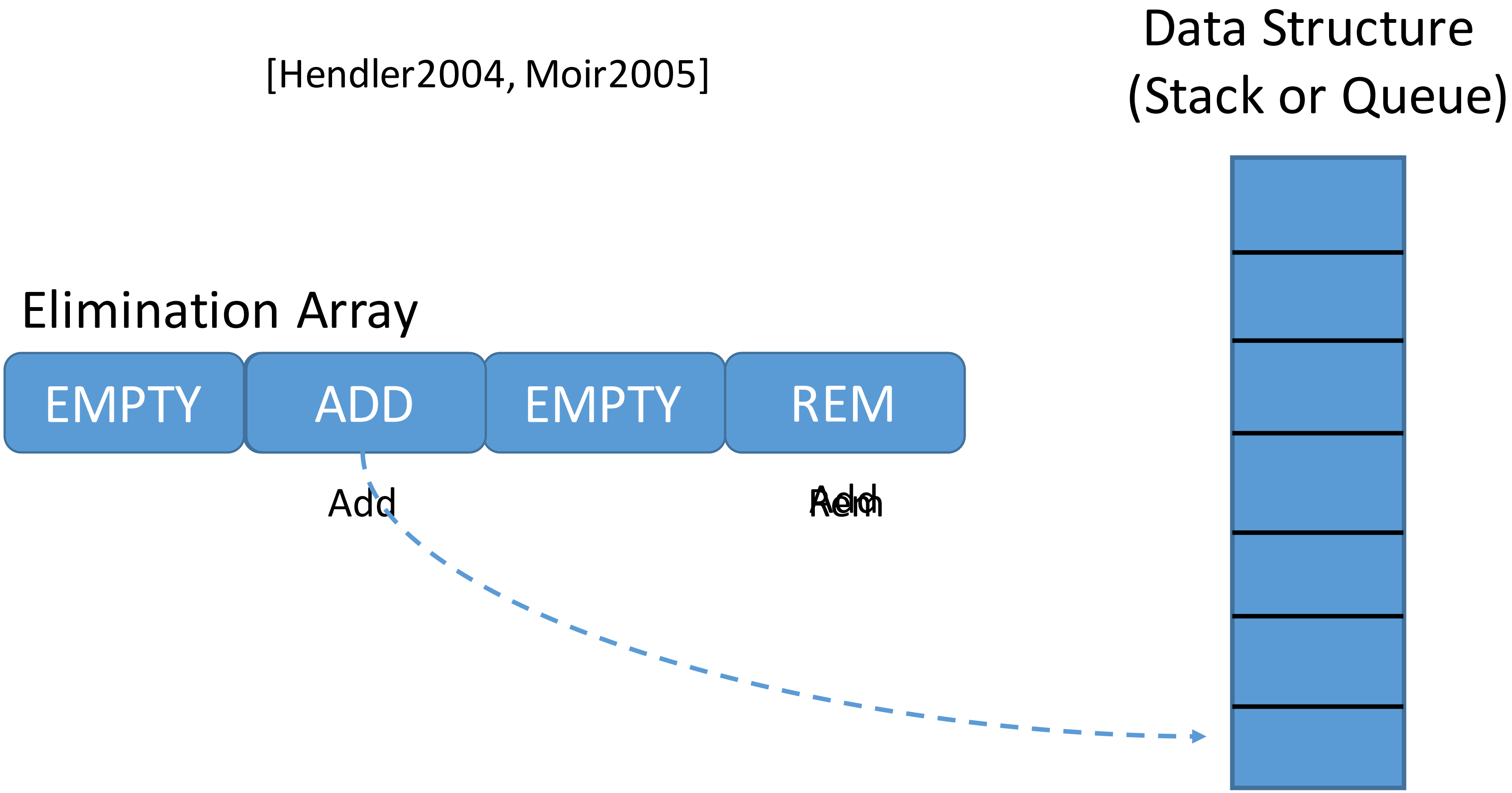
Tail
(Dummy)

Combined Removes
Parallel Adds ?

Parallel Adds

Stack and Queue Elimination (prior work)

[Hendler2004, Moir2005]



Data Structure
(Stack or Queue)

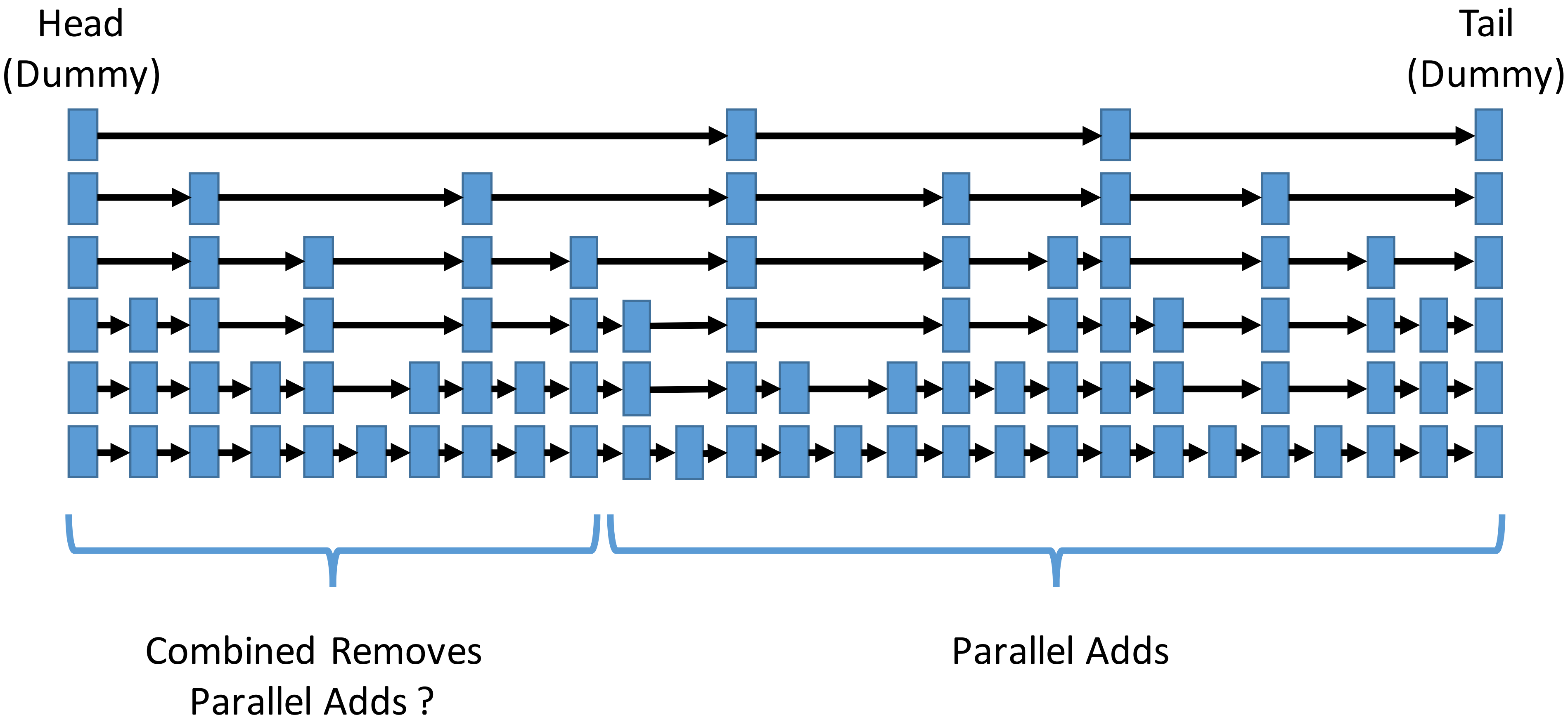
Elimination Array

EMPTY
ADD
EMPTY
REM

Add

Add
Rem

Parallelize Combined Adds Too: Use Elimination



Head
(Dummy)

Tail
(Dummy)

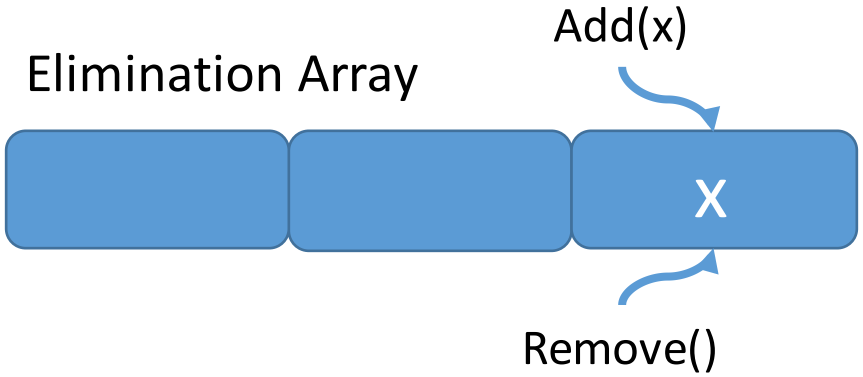
Combined Removes
Parallel Adds ?

Parallel Adds

The Priority Queue at a Glance

- Elimination
- RemoveMin and small-value Add combining
- Large-value Add parallelism

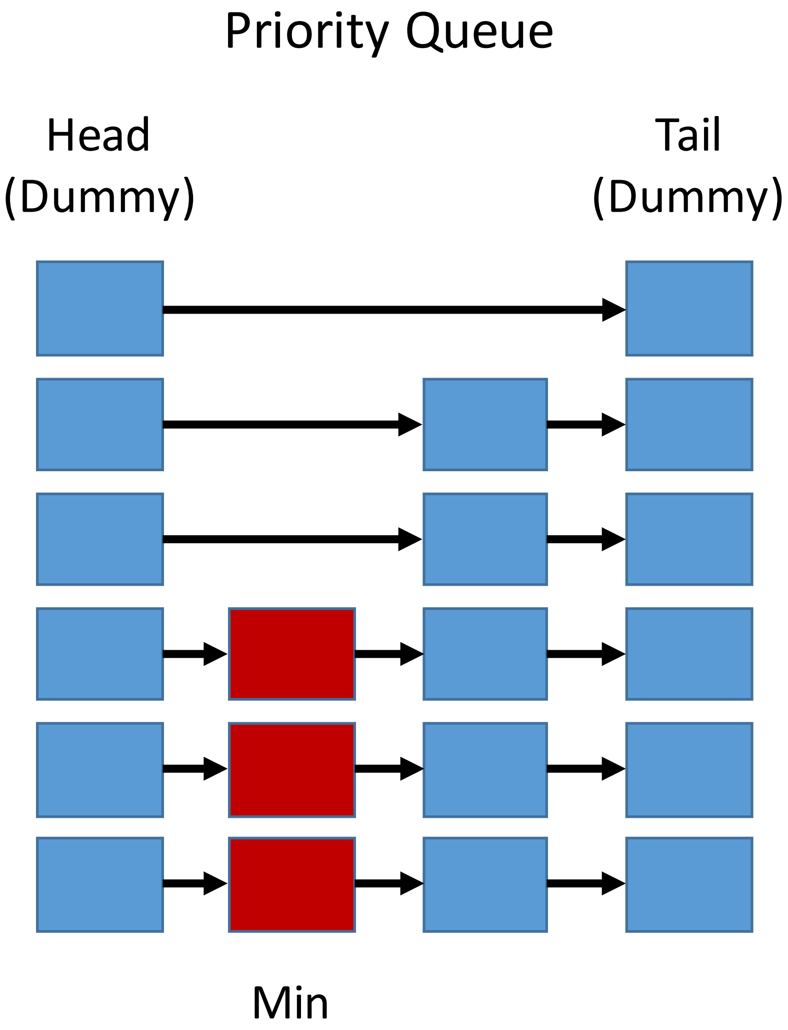
Implementation: Elimination



Elimination Array

Add(x)

Remove()



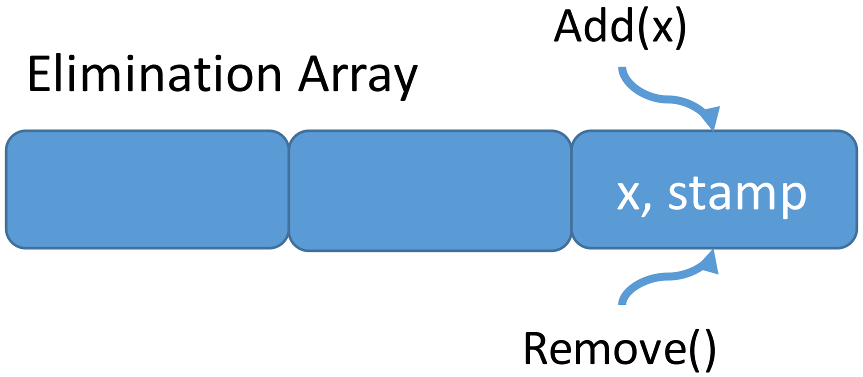
Priority Queue

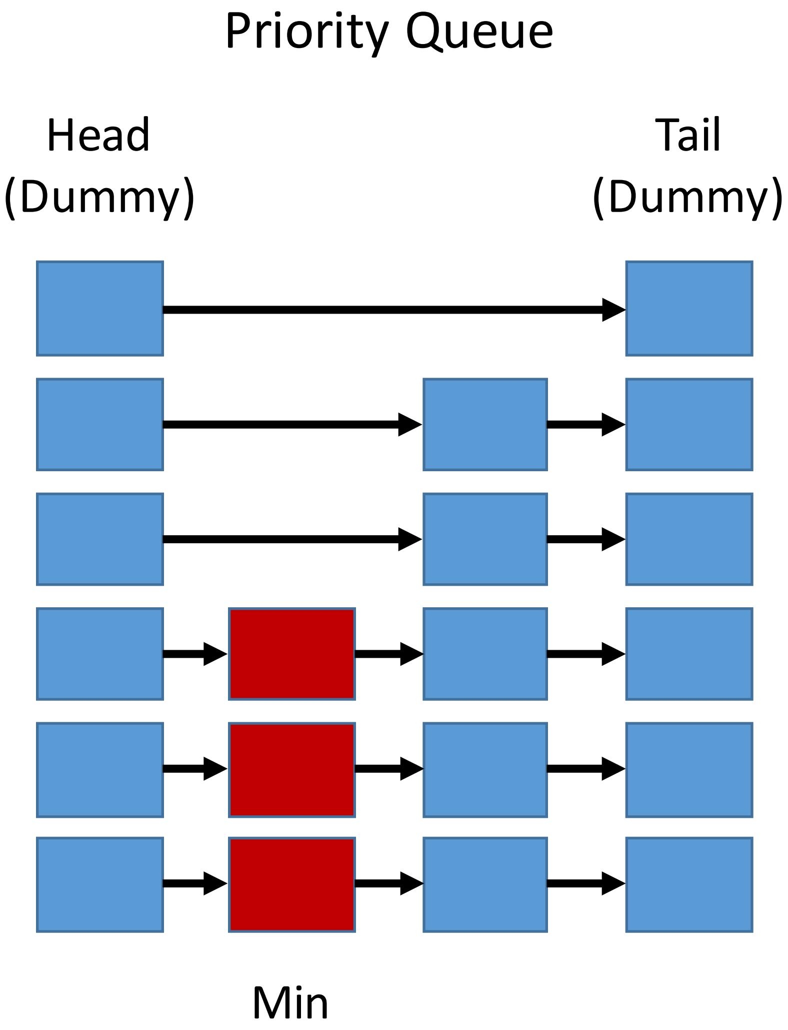
Head
(Dummy)

Tail
(Dummy)

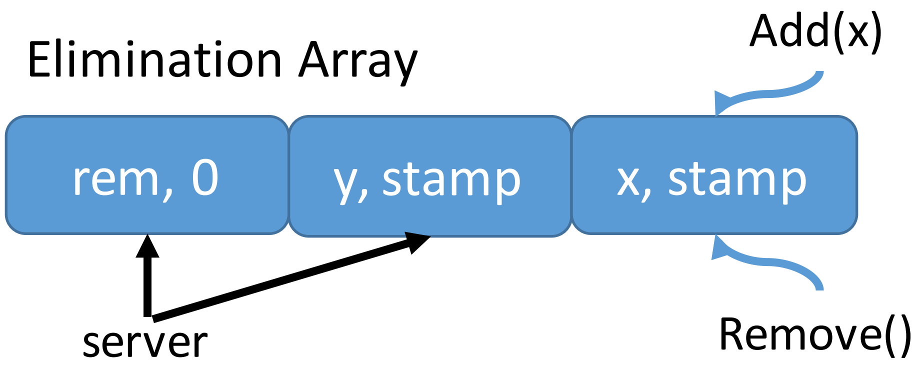
Min

Implementation: Elimination





Implementation: Combining



Elimination Array

Add(x)

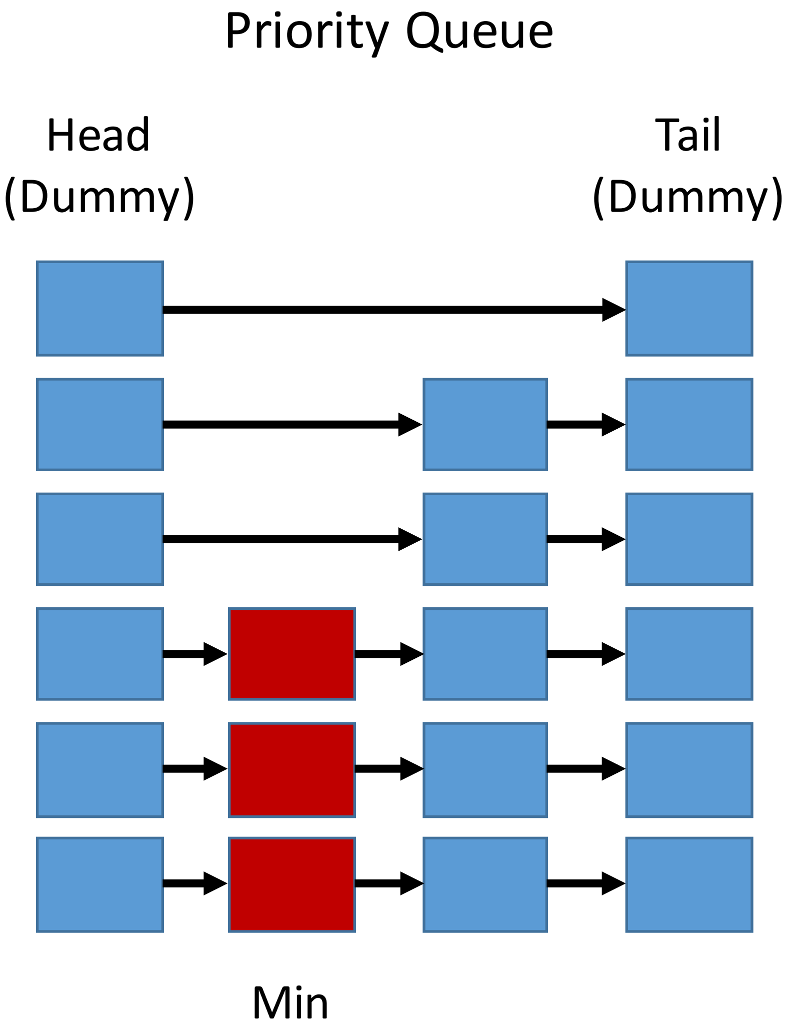
rem, 0

y, stamp

x, stamp

server

Remove()



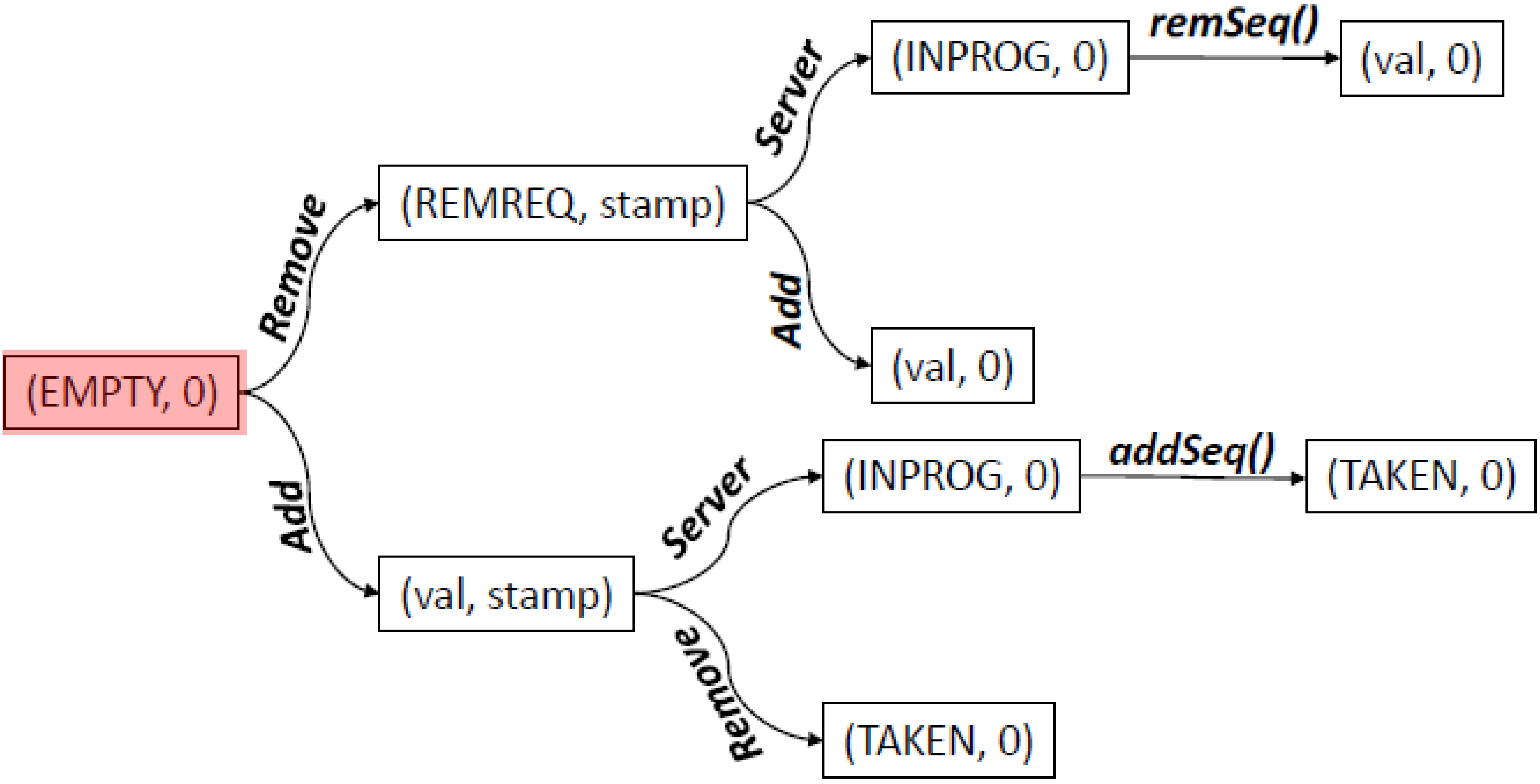
Priority Queue

Head
(Dummy)

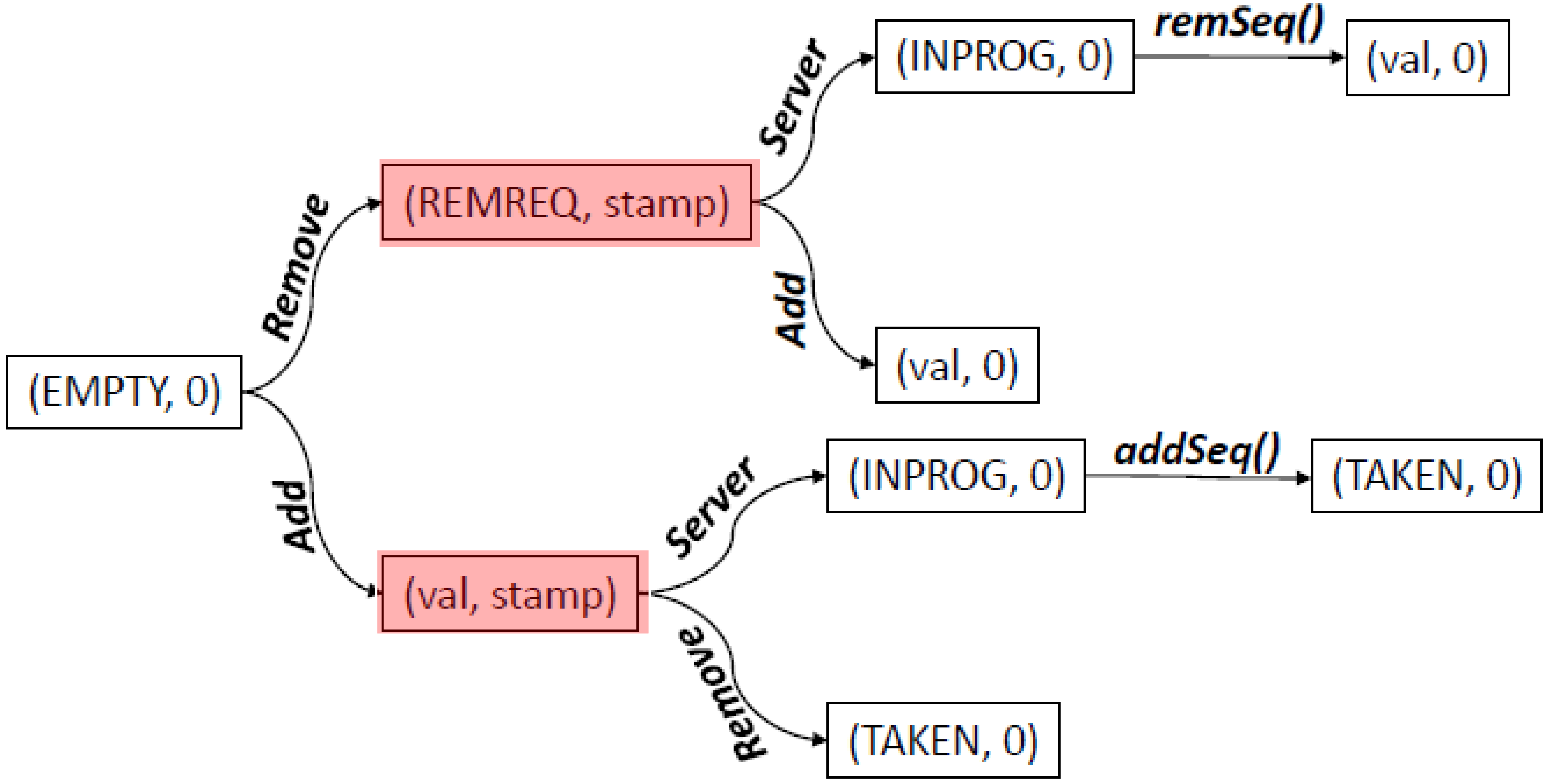
Tail
(Dummy)

Min

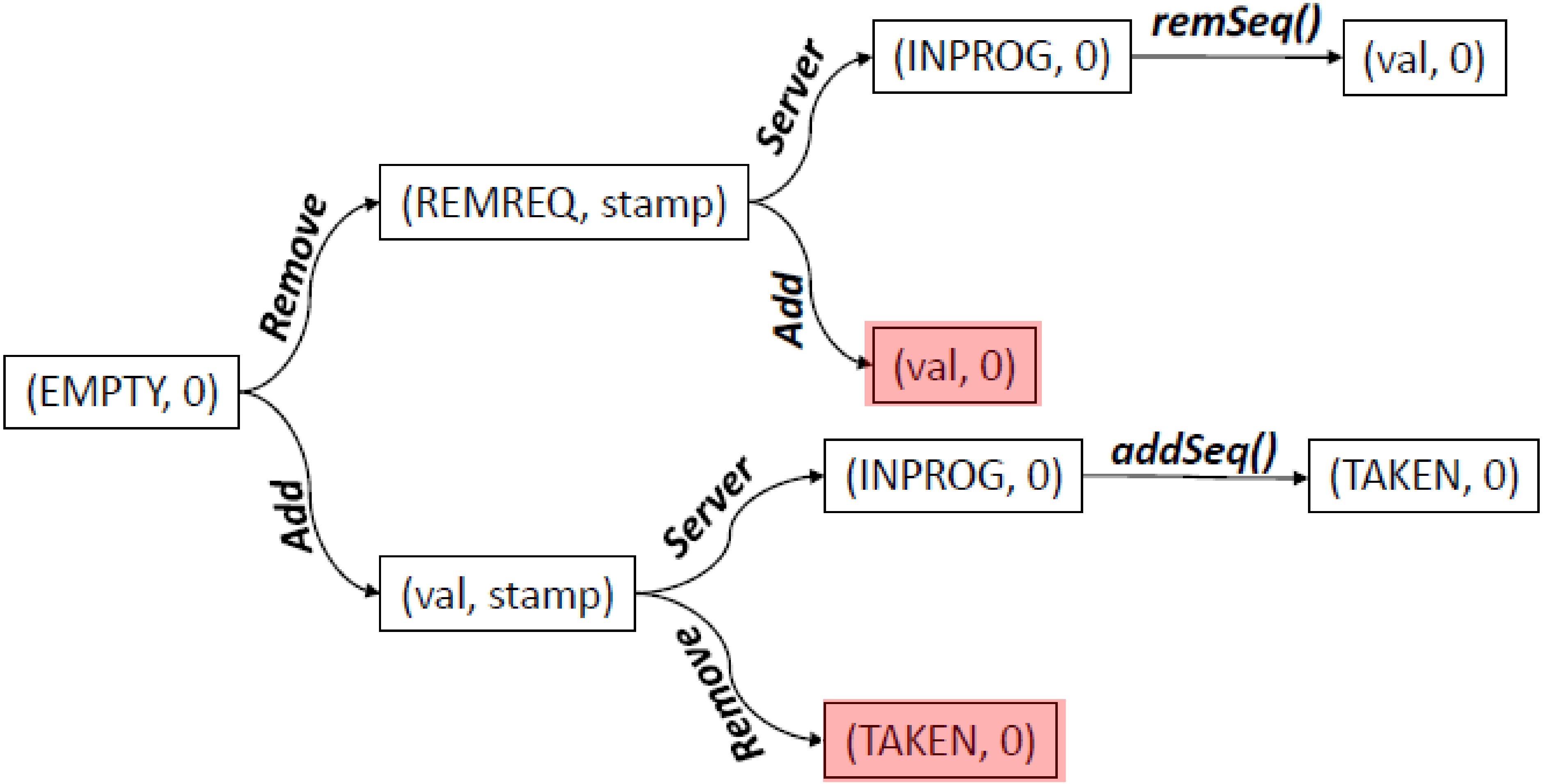
Transitions of a Slot in the Elimination Array



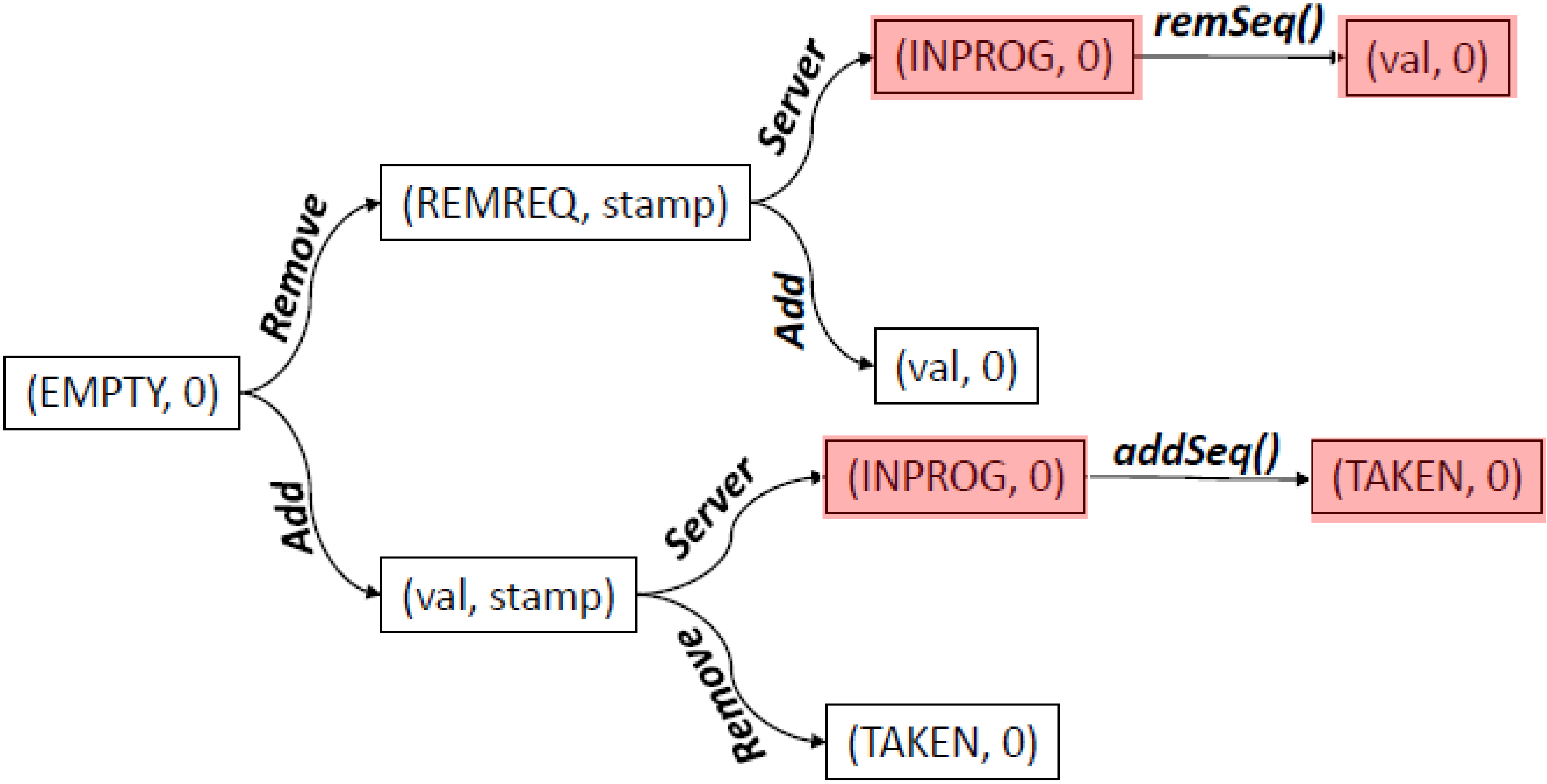
Transitions of a Slot in the Elimination Array



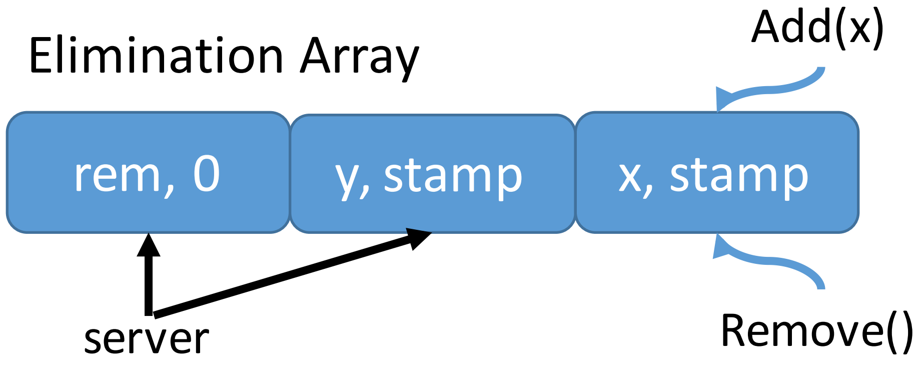
Transitions of a Slot in the Elimination Array



Transitions of a Slot in the Elimination Array



Implementation: Parallel Adds



Elimination Array

Add(x)

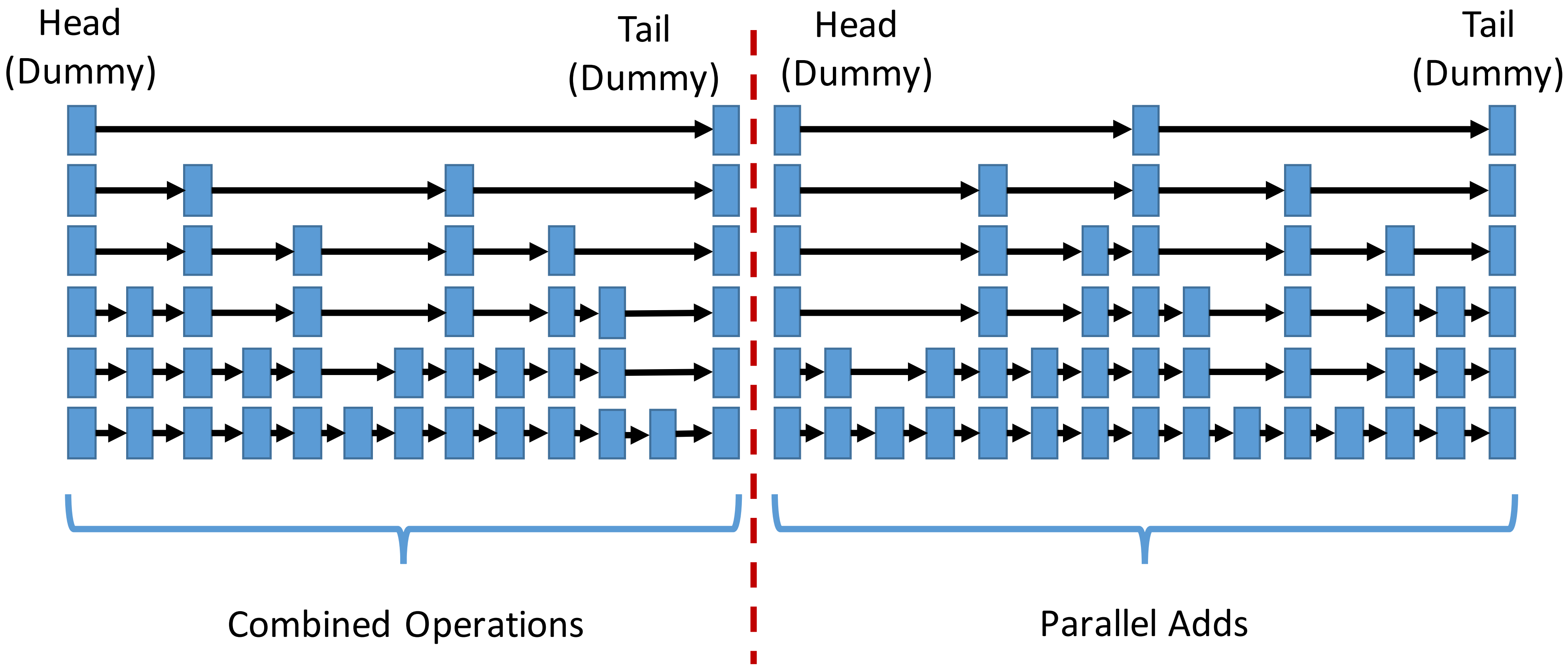
rem, 0

y, stamp

x, stamp

server

Remove()



Head
(Dummy)

Tail
(Dummy)

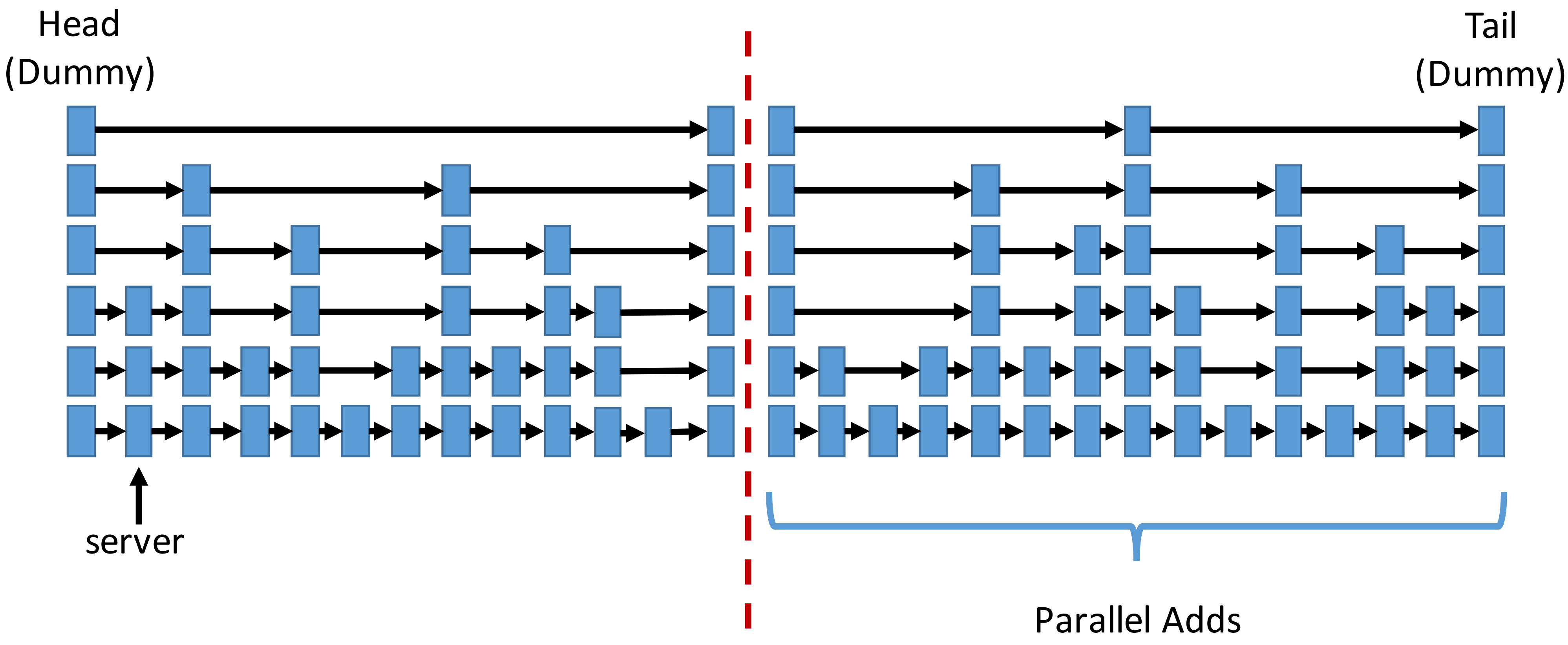
Head
(Dummy)

Tail
(Dummy)

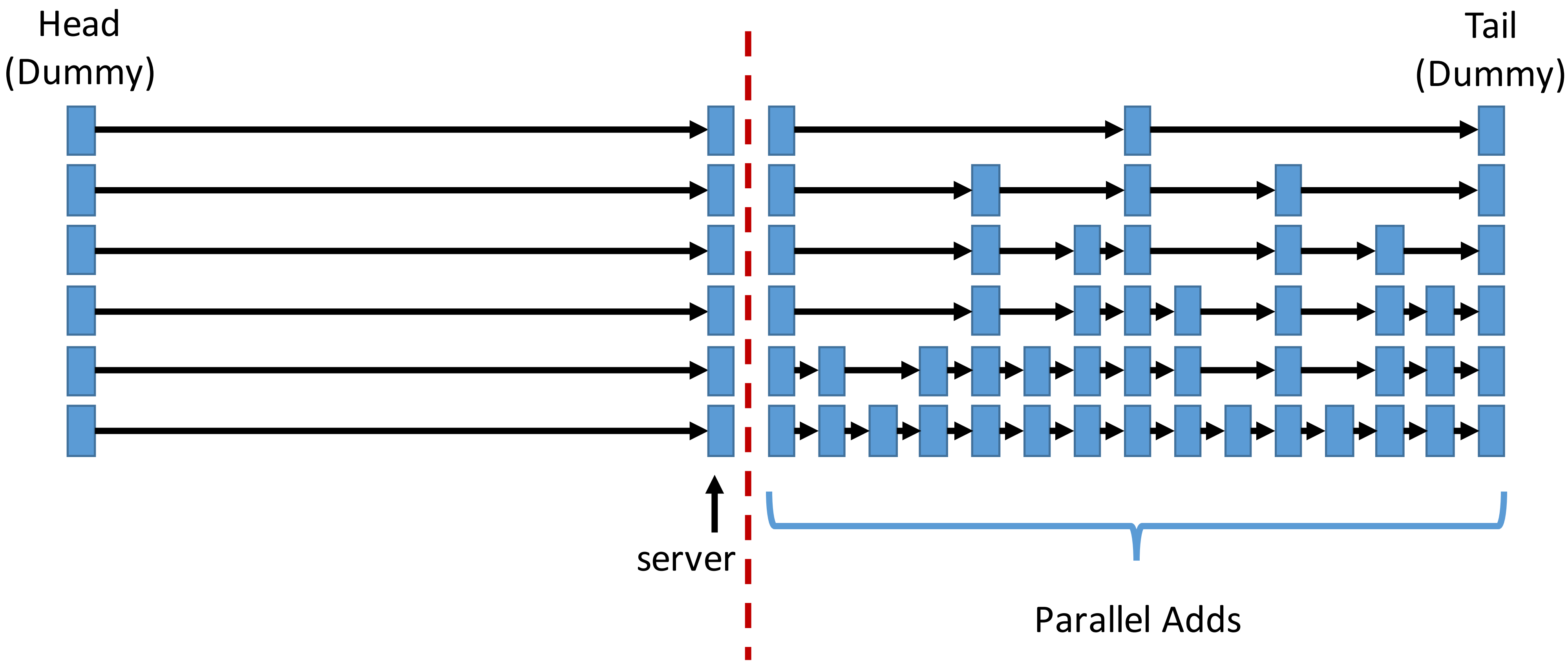
Combined Operations

Parallel Adds

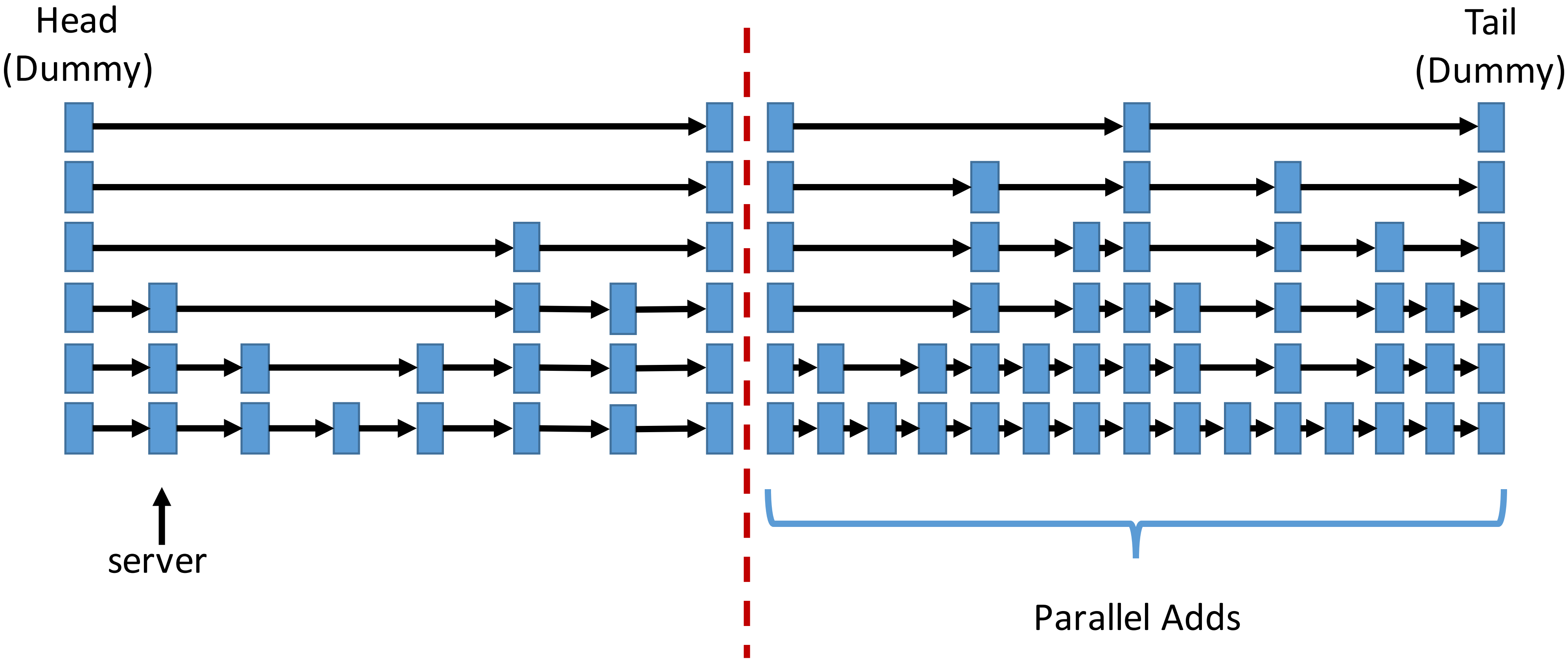
Adaptive PQ Split: moveHead()



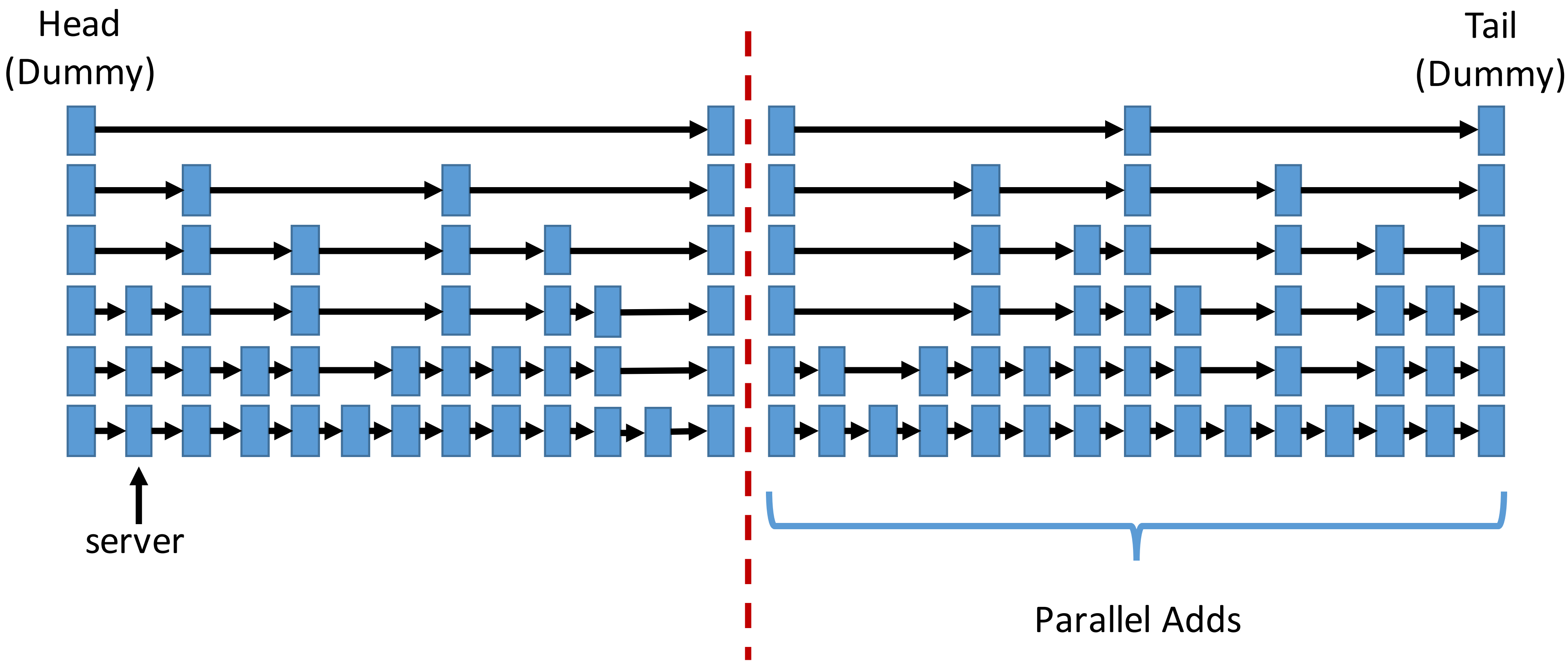
Adaptive PQ Split: moveHead()



Adaptive PQ Split: chopHead()



Adaptive PQ Split: chopHead()



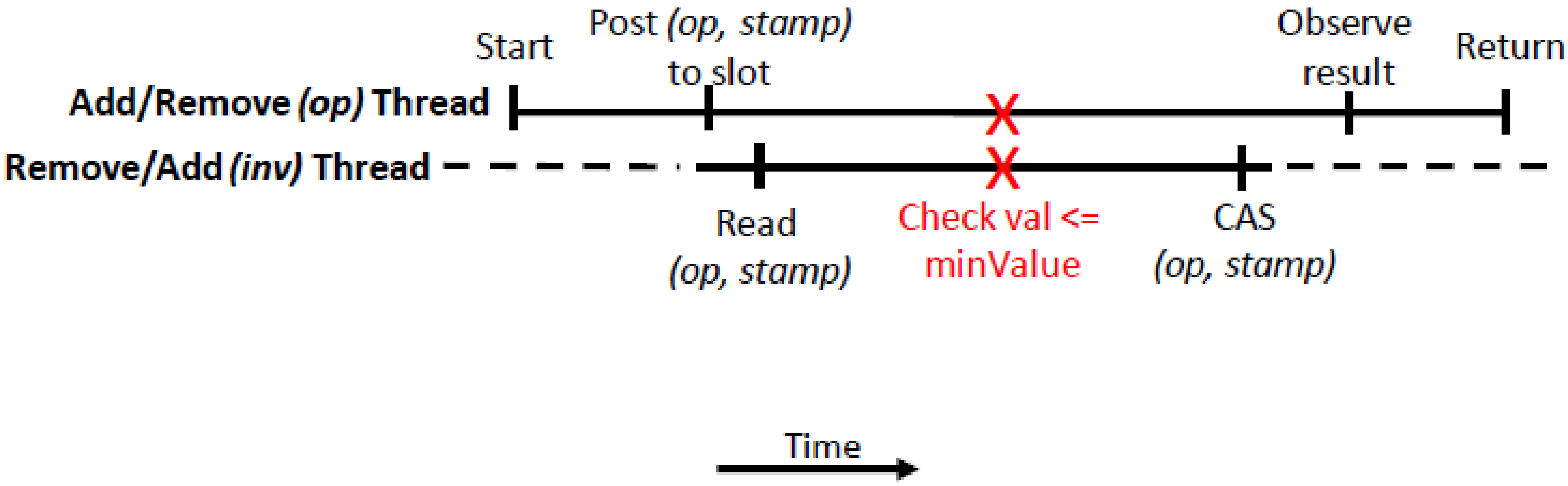
Synchronization

- MoveHead() and chopHead() change the parallel skiplist
- We need to synchronize server and parallel adds
- Use RW Lock
- Server: acquire writeLock for moveHead() and chopHead()
- Parallel adds: acquire readLock

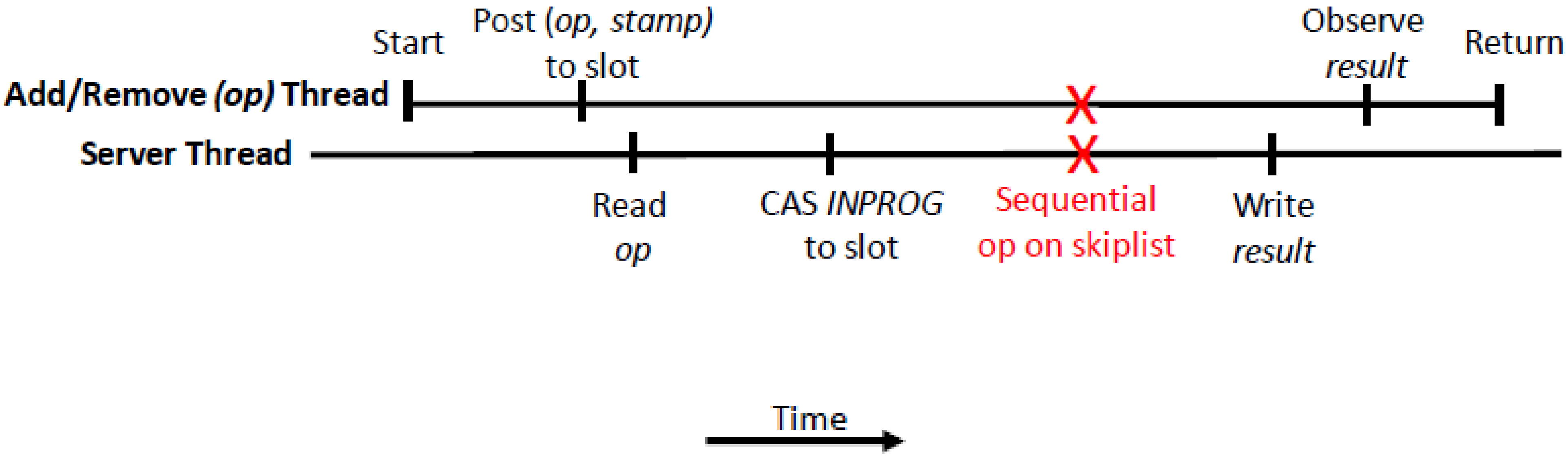
Synchronization

- Single writer lock
- Writer preference
- Implementation: based on timestamps
    - Server increments timestamp for moveHead() and chopHead()
    - Don’t hold the lock for the whole time of the parallel add
    - Do a clean find first (as verified by the timestamp)
    - Acquire read lock and finish the insertion

Linearizability - Elimination



Linearizability - Combining



Add/Remove *(op)* Thread

Start

Post *(op, stamp)* to slot

Observe *result*

Return

Server Thread

Read *op*

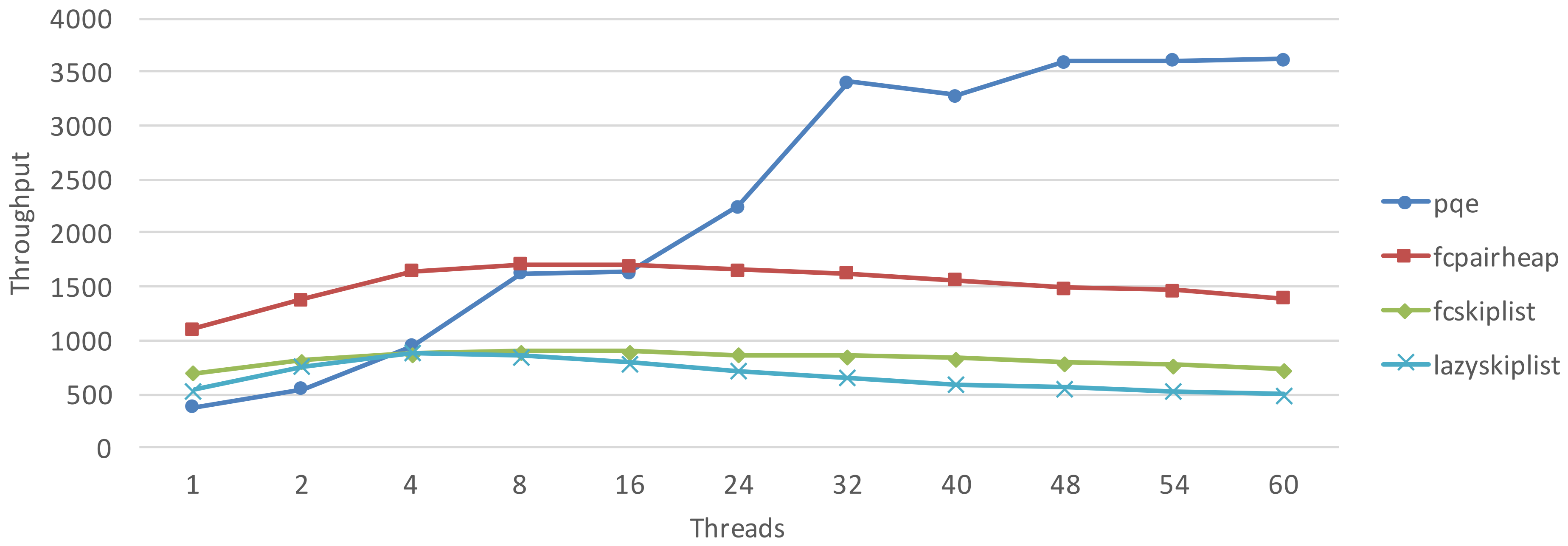
CAS *INPROG* to slot

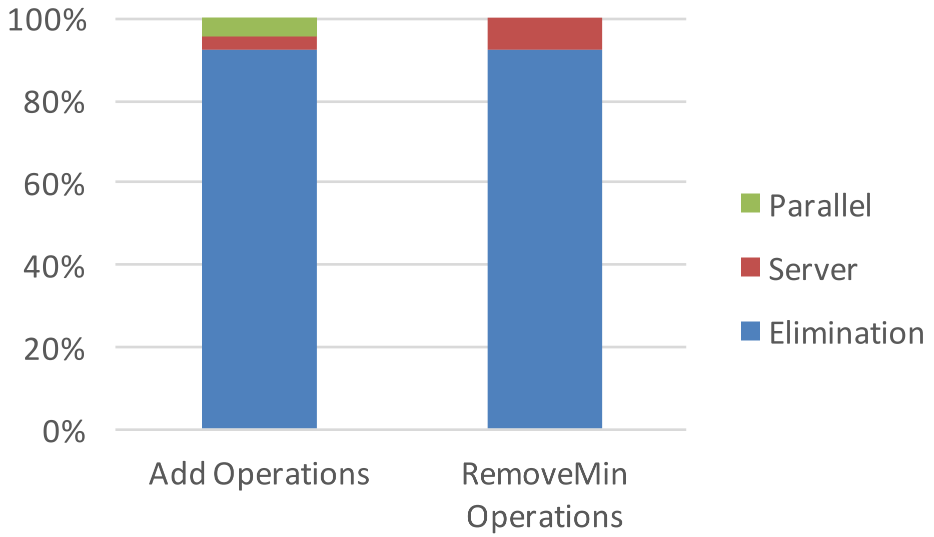
Sequential op on skiplist

Write *result*

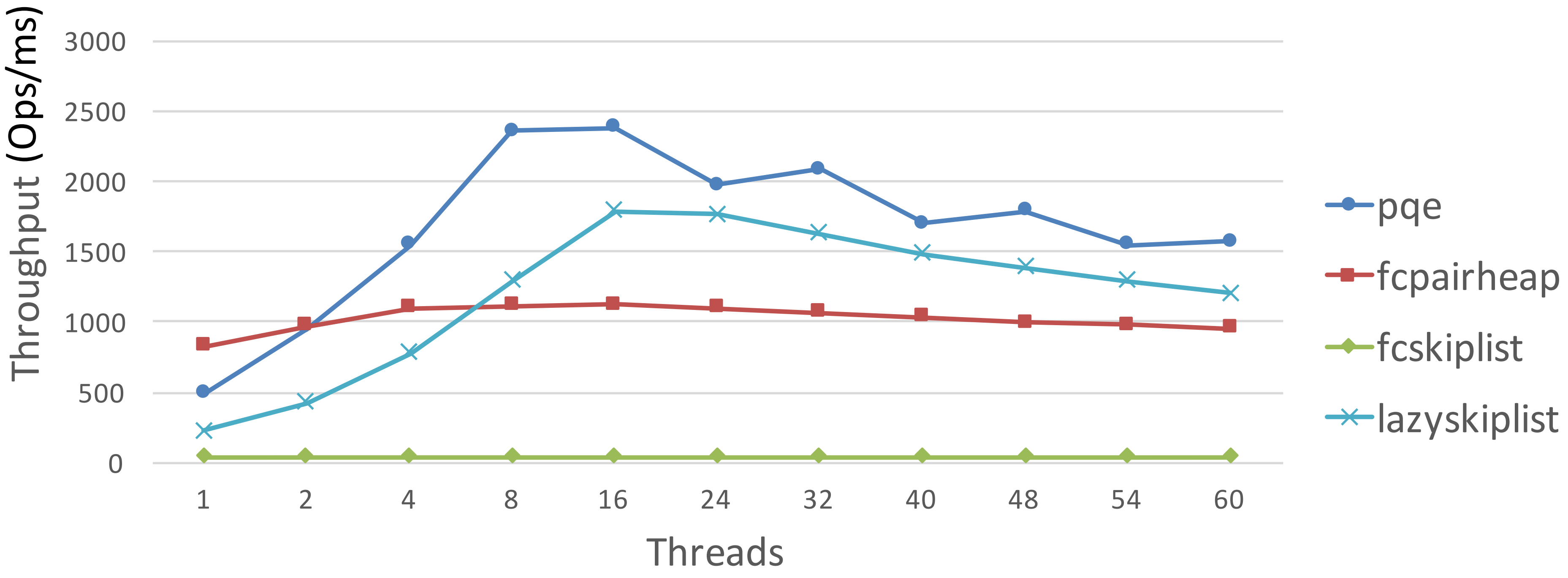
Time

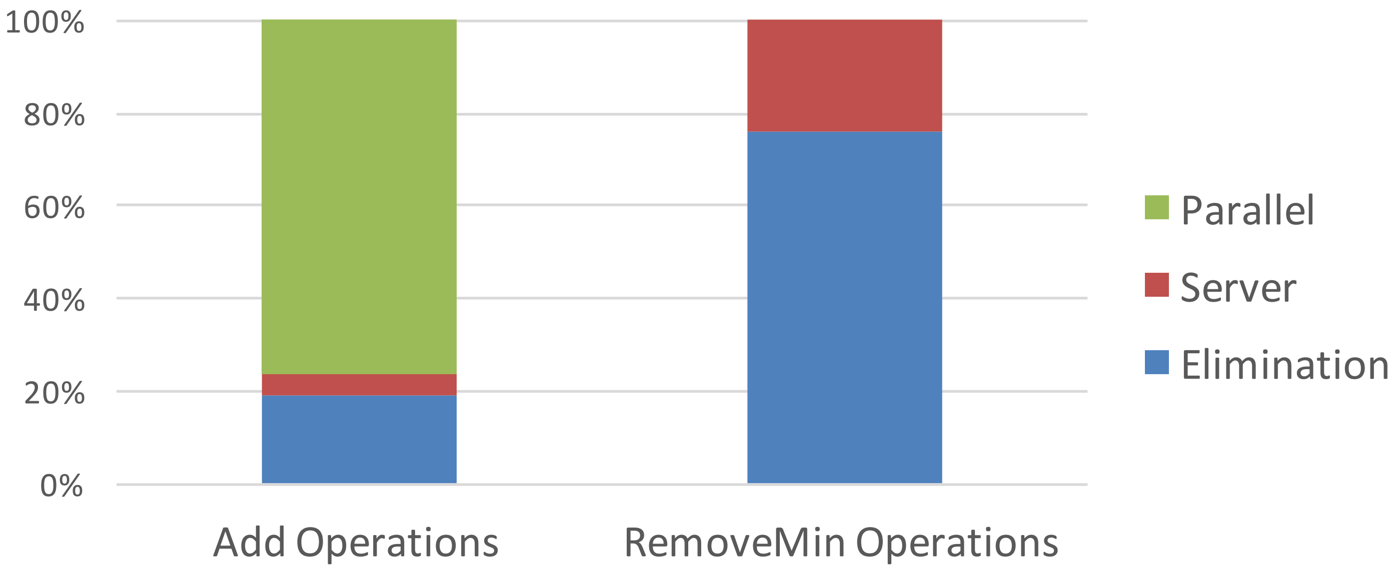
50% Add Operations 50% RemoveMin Operations



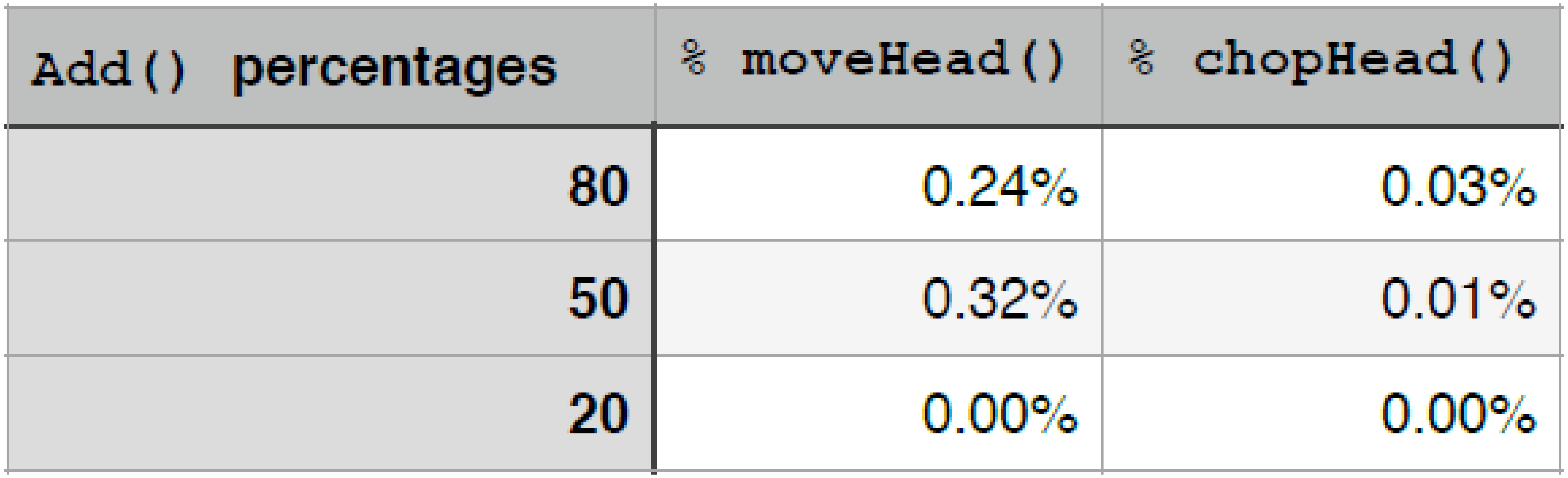


80% Add Operations 20% RemoveMin Operations





Impact of Maintaining Two Skiplists



Add() percentages	% moveHead()	% chopHead()
80	0.24%	0.03%
50	0.32%	0.01%
20	0.00%	0.00%

Hardware Transactions - Motivation

- RW Lock can be too expensive
- Use hardware transactions
- Intel TSX
- Speculative execution

Hardware Transactions (1)

- Naïve version

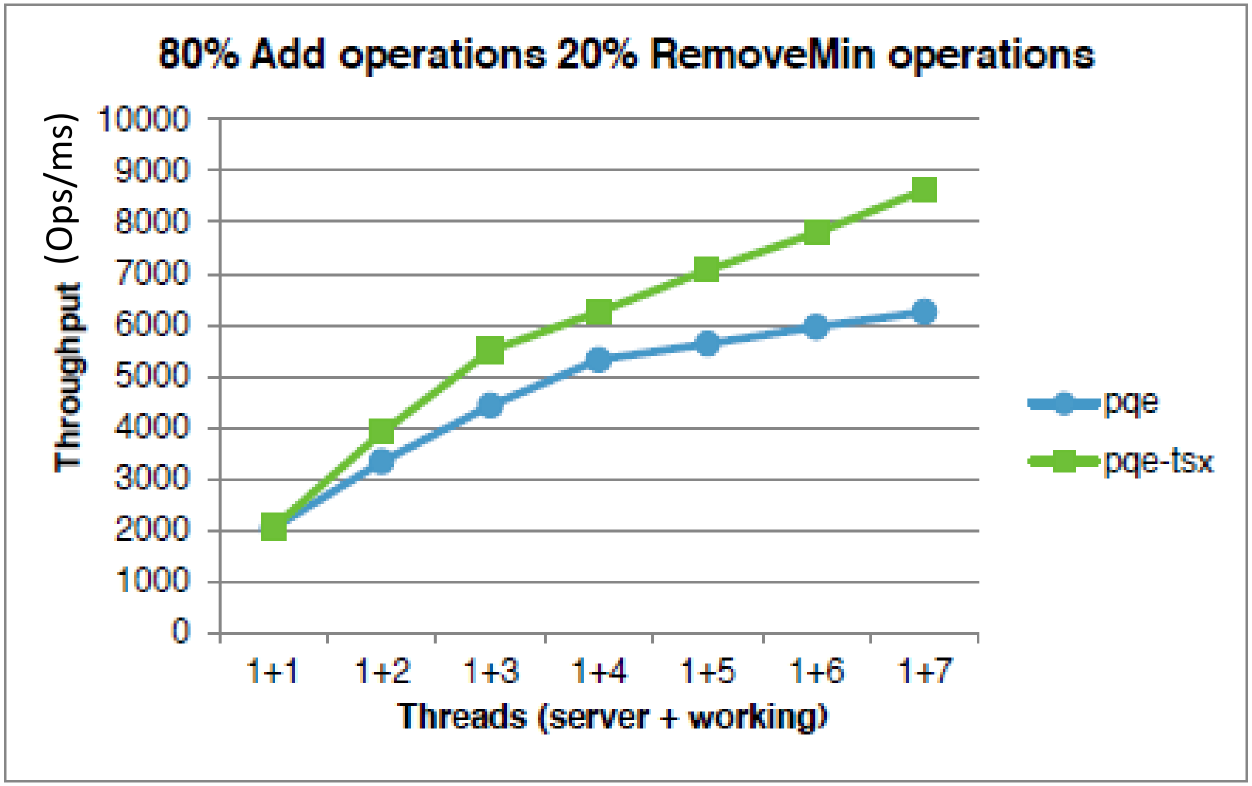
- Start a transaction
- Find + Insert
- End transaction

- Too many aborts

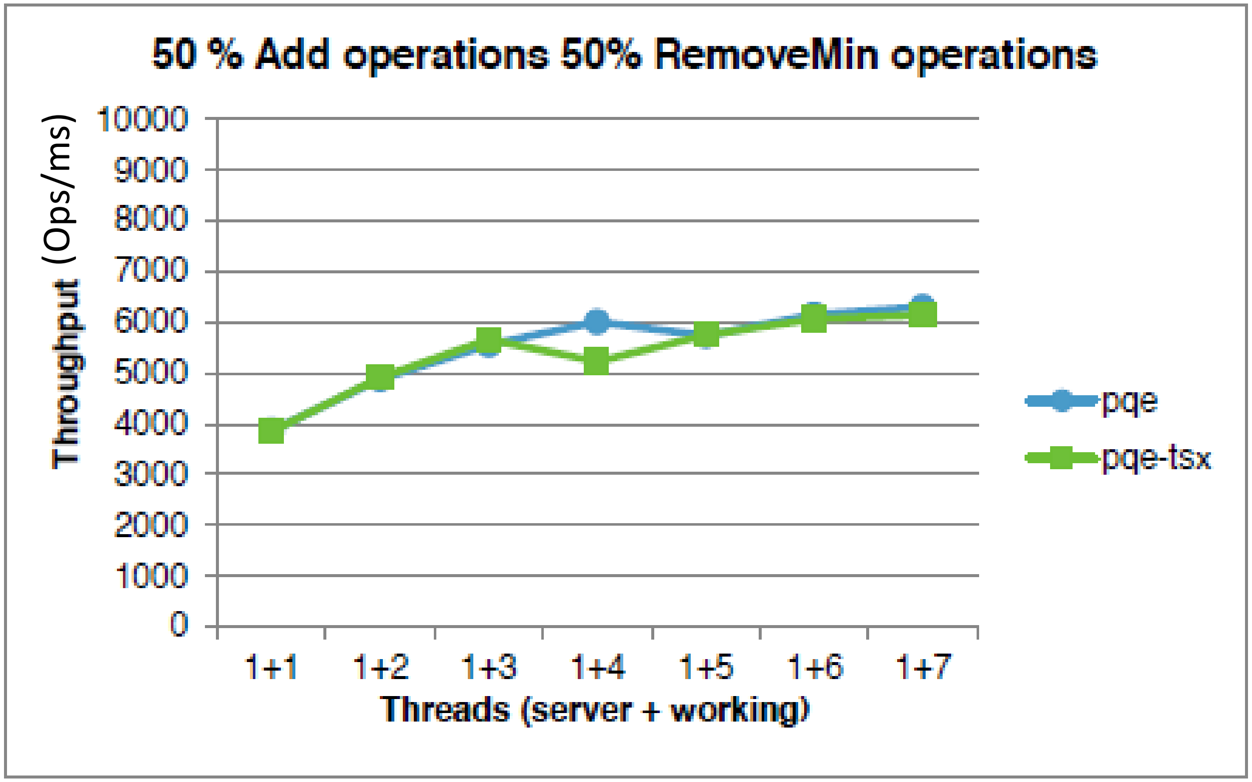
Hardware Transactions (2)

- Timestamp approach
- Server increments timestamp for moveHead() and chopHead()
- Find executes non-transactionally but has to be restarted if timestamp changes
- Insert executed in a transaction
- Read the timestamp in the transaction

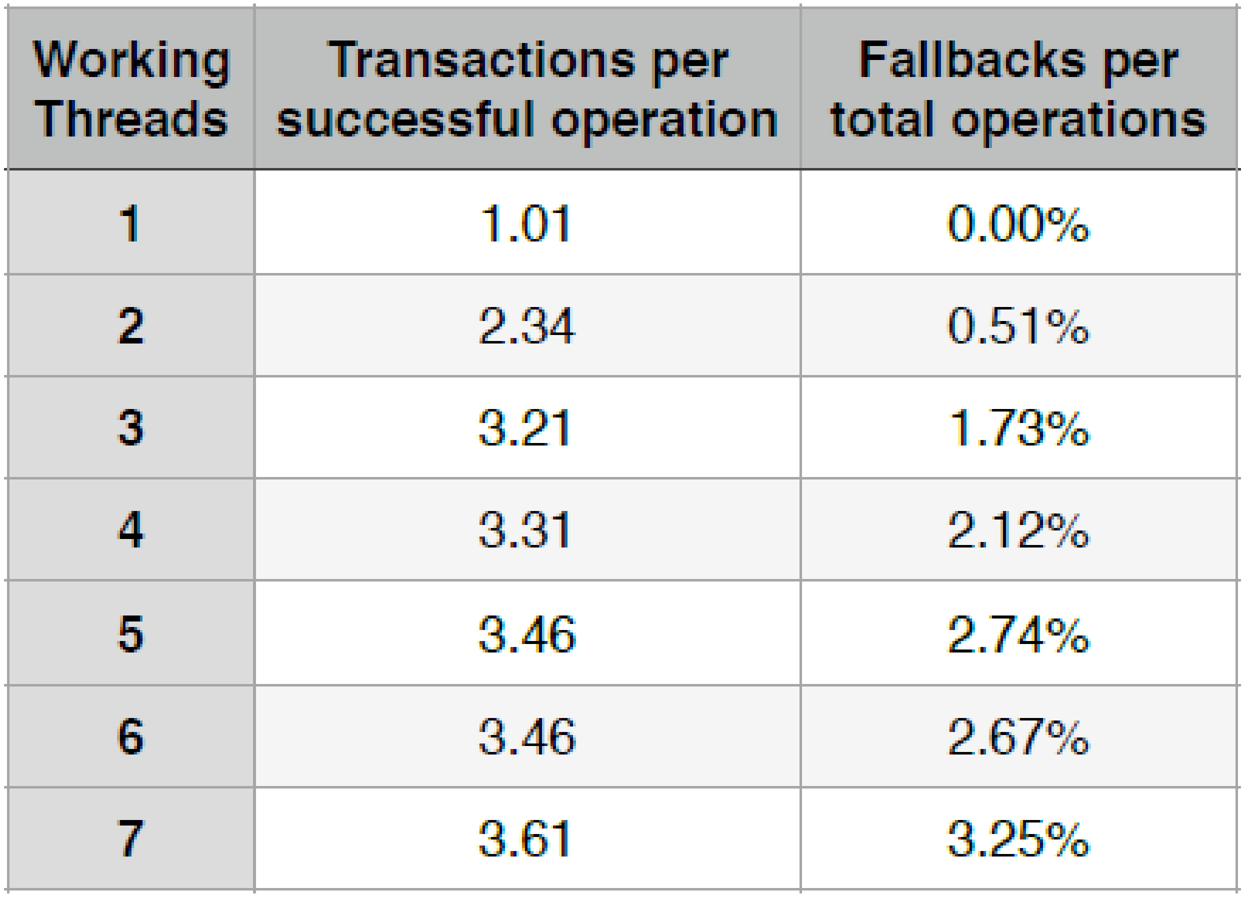
Using Hardware Transactions



Using Hardware Transactions



Transactions Stats for 50% Add() and 50% RemoveMin()



Working Threads	Transactions per successful operation	Fallbacks per total operations
1	1.01	0.00%
2	2.34	0.51%
3	3.21	1.73%
4	3.31	2.12%
5	3.46	2.74%
6	3.46	2.67%
7	3.61	3.25%

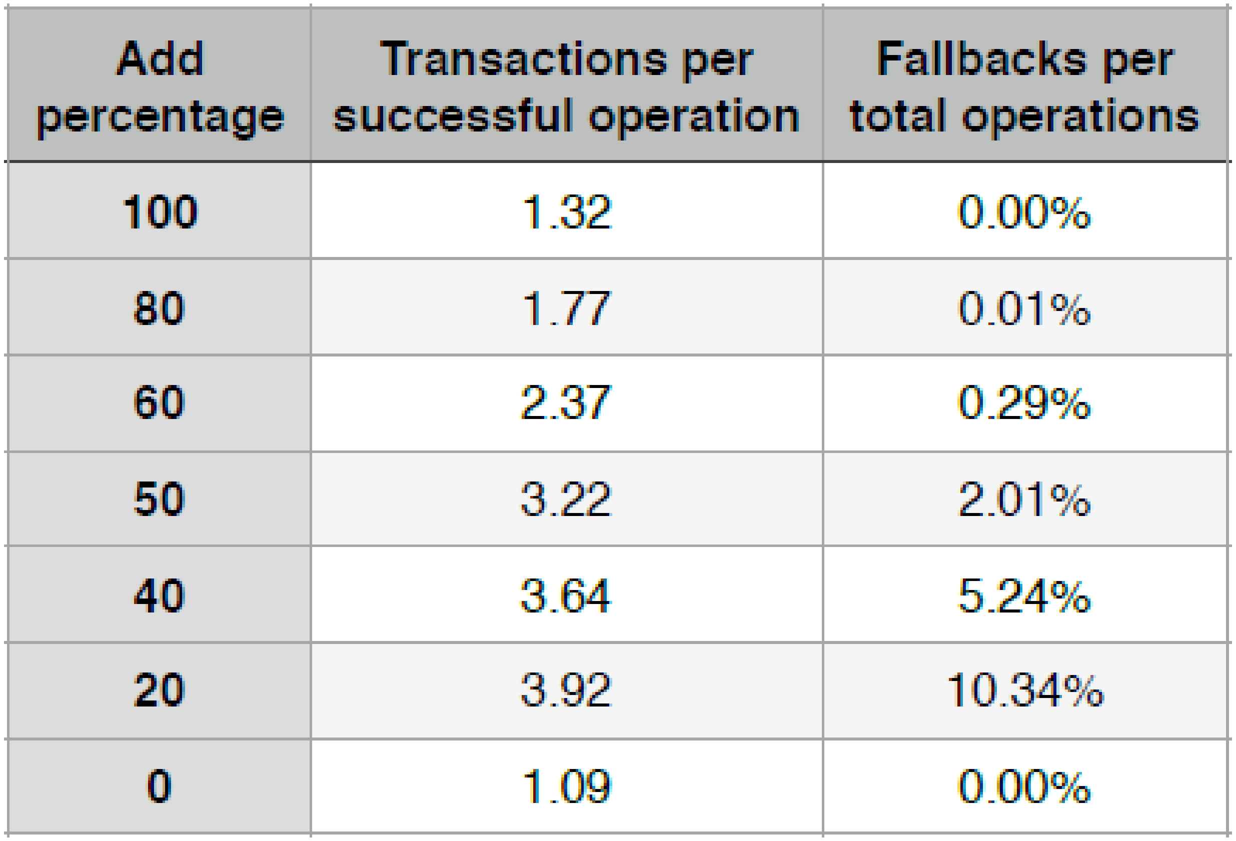
Summary

- First elimination algorithm for a priority queue
- Use two skiplist to separate small adds from large value adds
- Combining + Parallel Adds + Elimination
- HTM simplified the algorithm and improved performance

cs.brown.edu/~irina

cs.brown.edu/~hmendes

Transactions Stats for 3 Working Threads, 1 Server Thread



Add percentage	Transactions per successful operation	Fallbacks per total operations
100	1.32	0.00%
80	1.77	0.01%
60	2.37	0.29%
50	3.22	2.01%
40	3.64	5.24%
20	3.92	10.34%
0	1.09	0.00%