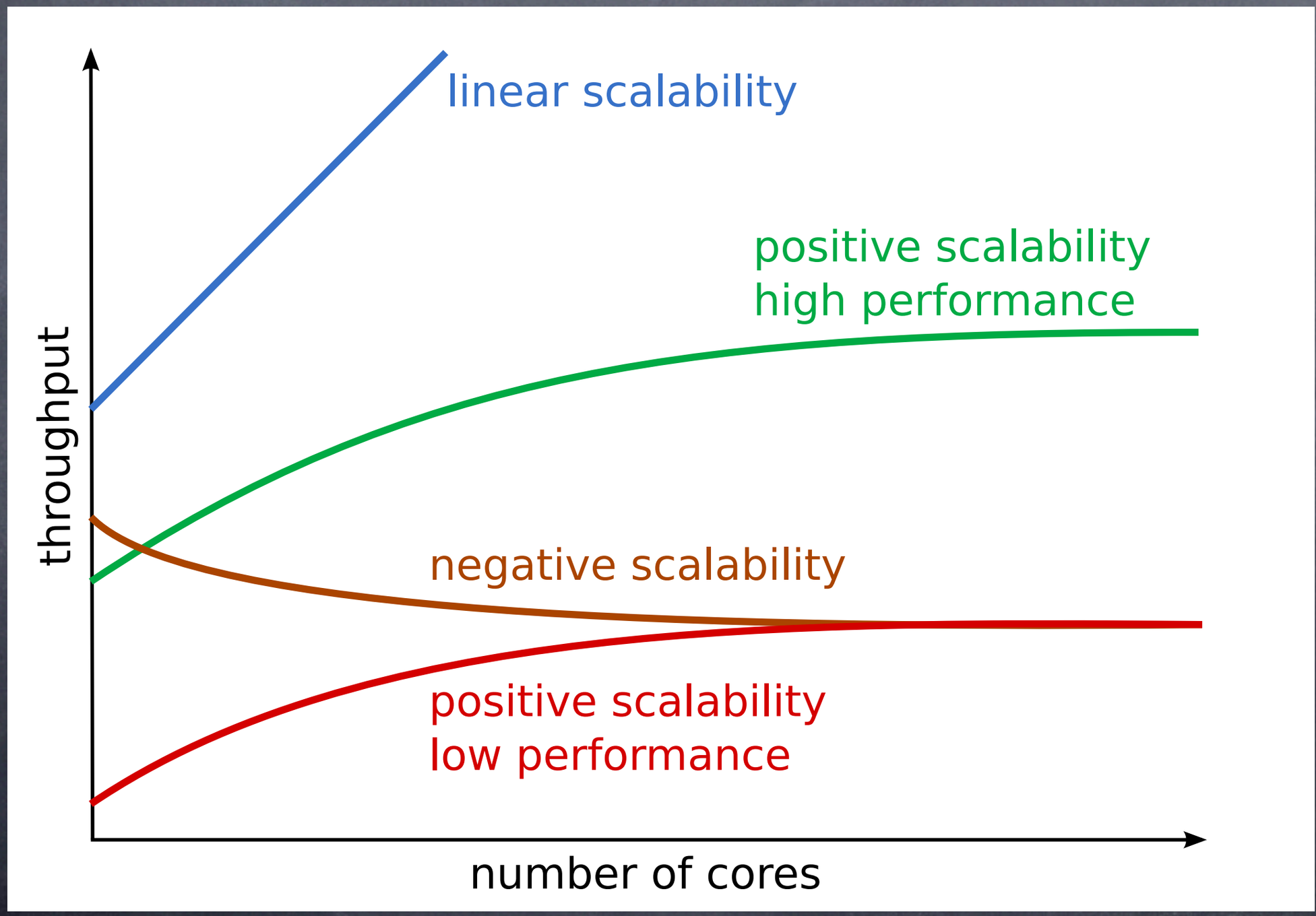
Performance, Scalability, and Semantics of Concurrent FIFO Queues

Christoph Kirsch, Hannes Payer,
Harald Röck, Ana Sokolova

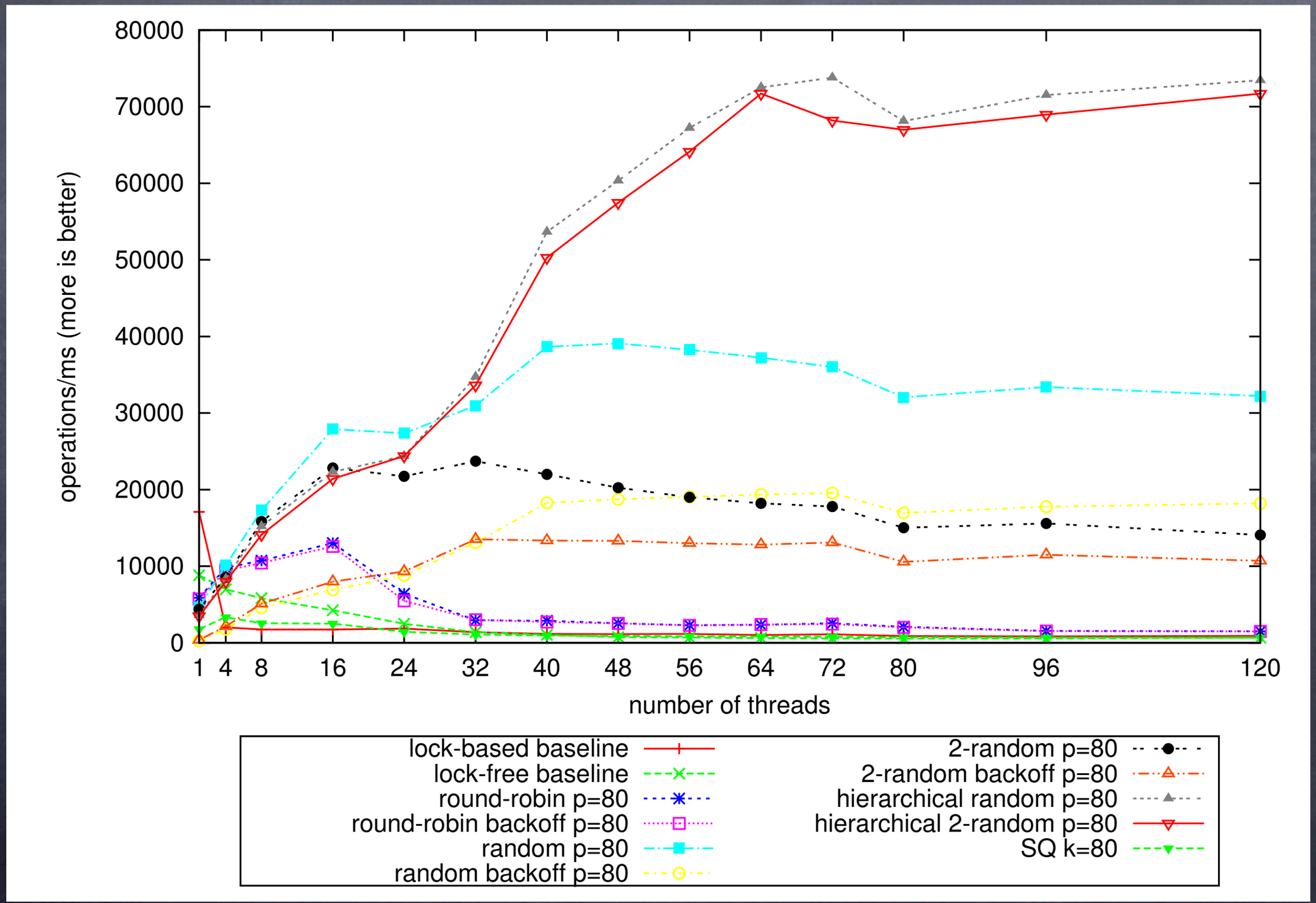
Universität Salzburg

Advanced Operating Systems Class, January 2012

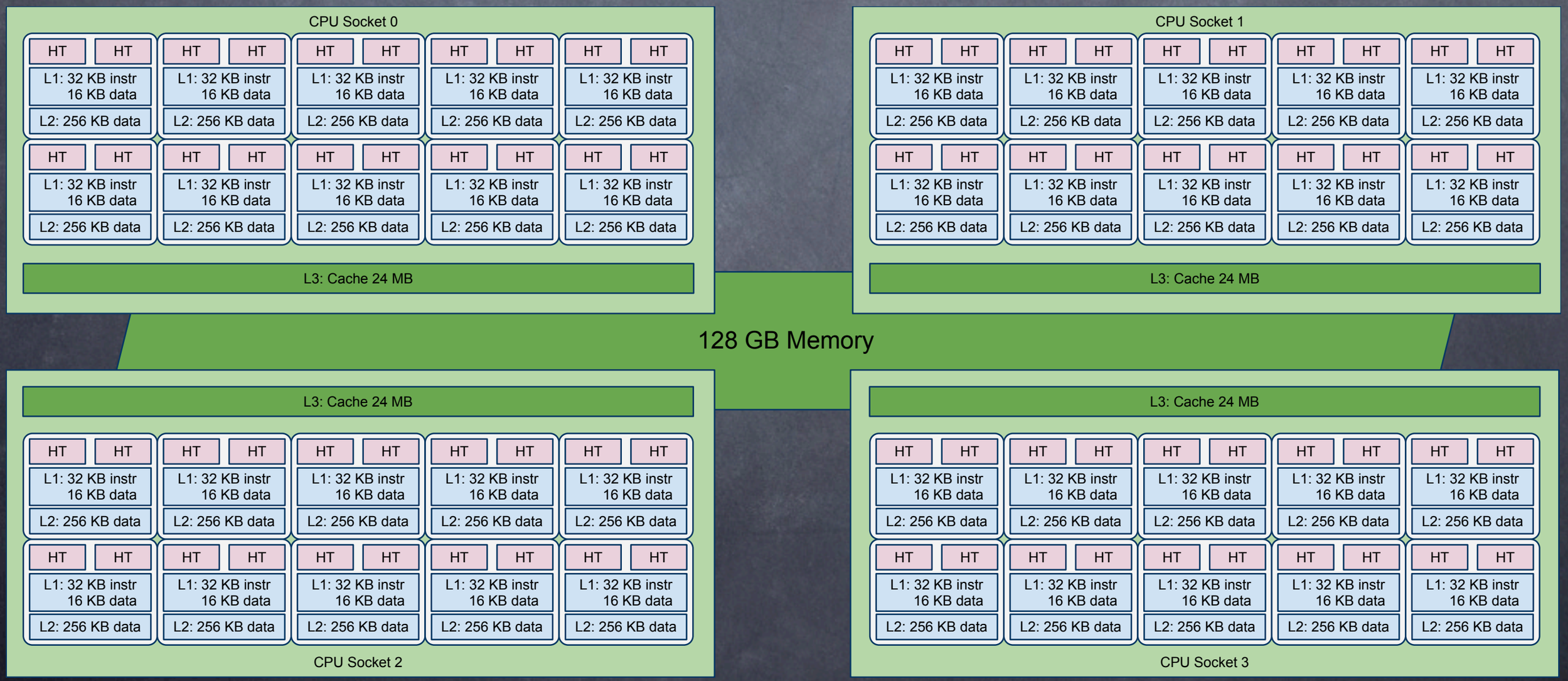
Performance & Scalability



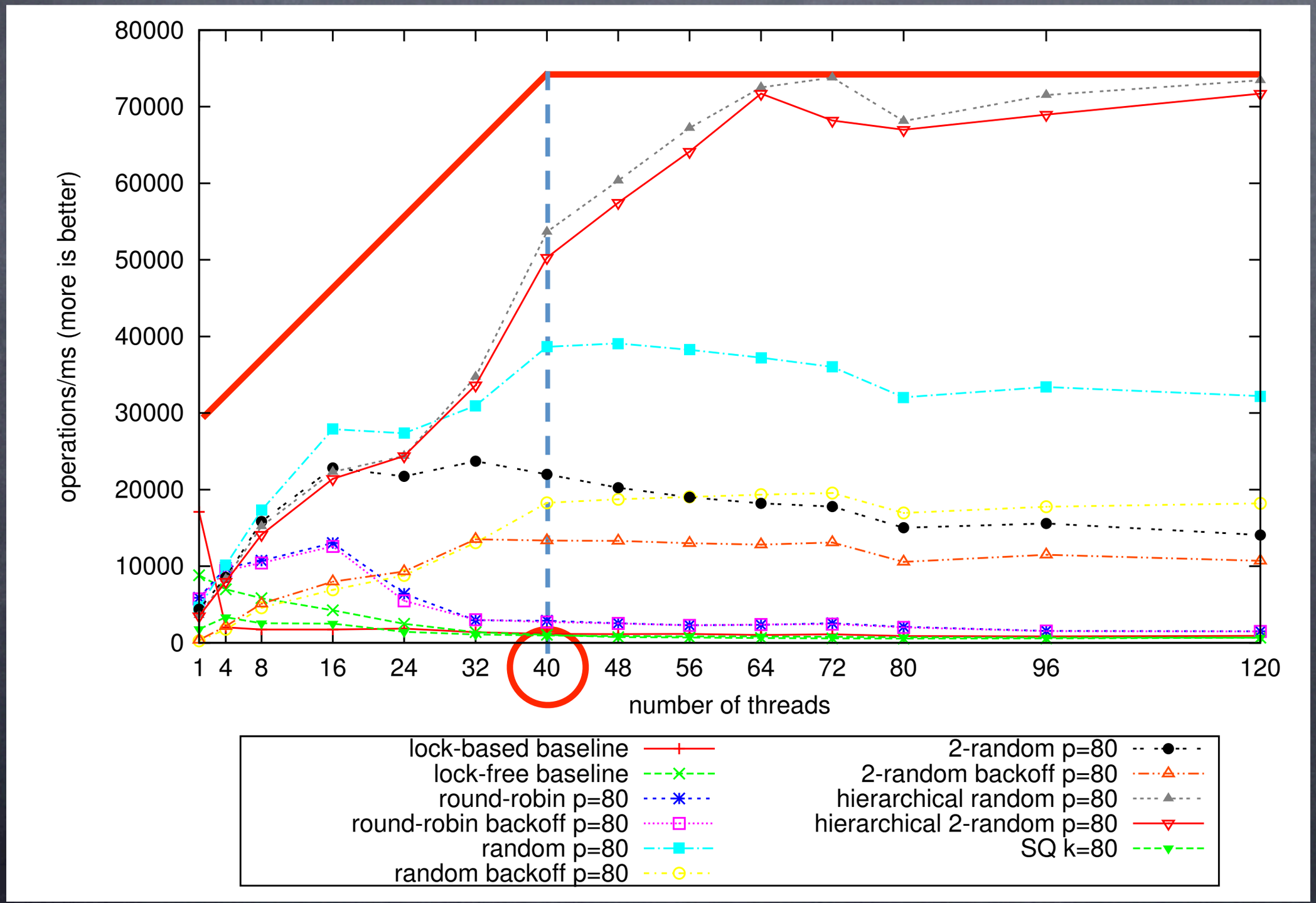
High Contention



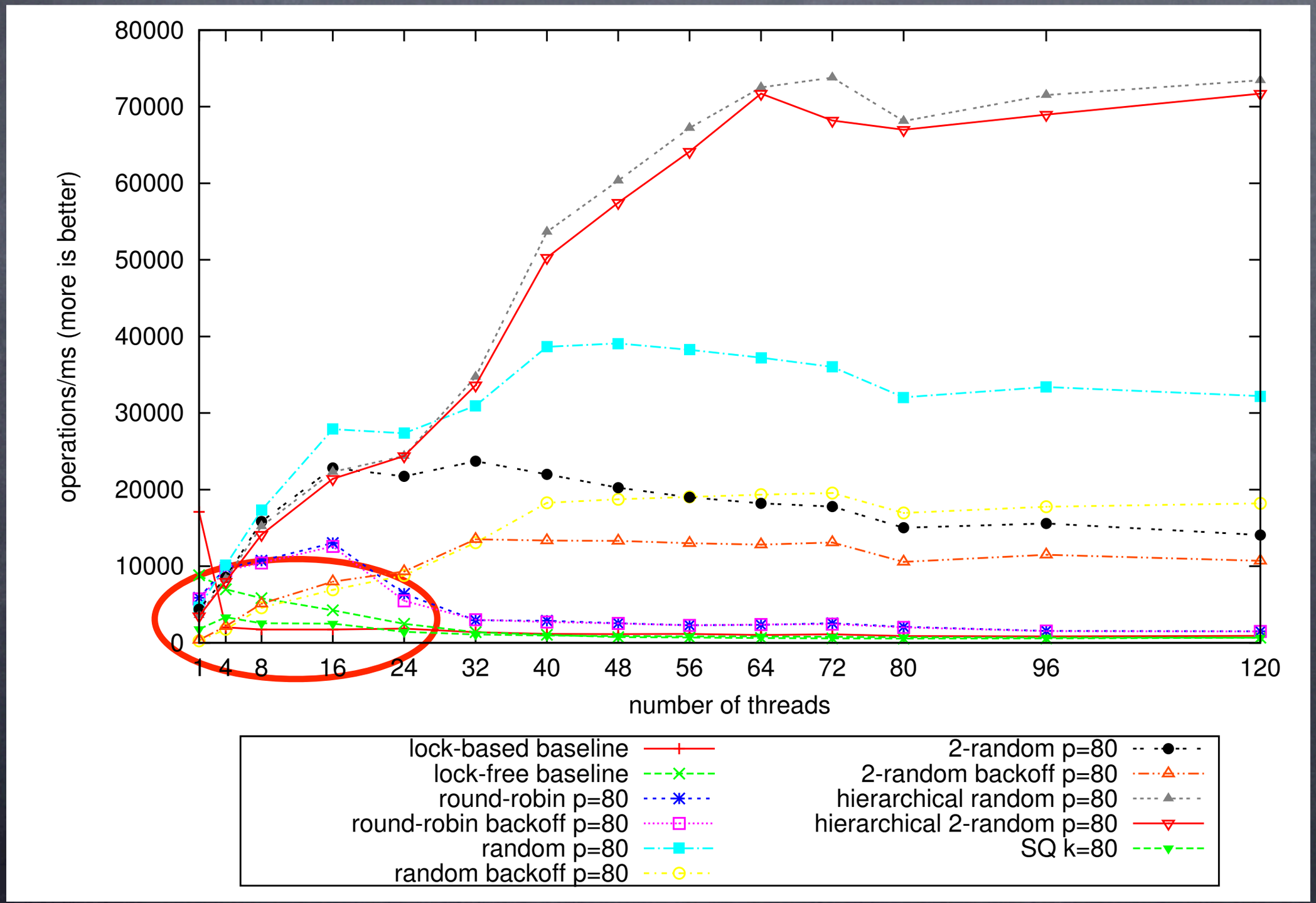
4 processors x 10 cores x 2 hardware threads = 80 hardware threads



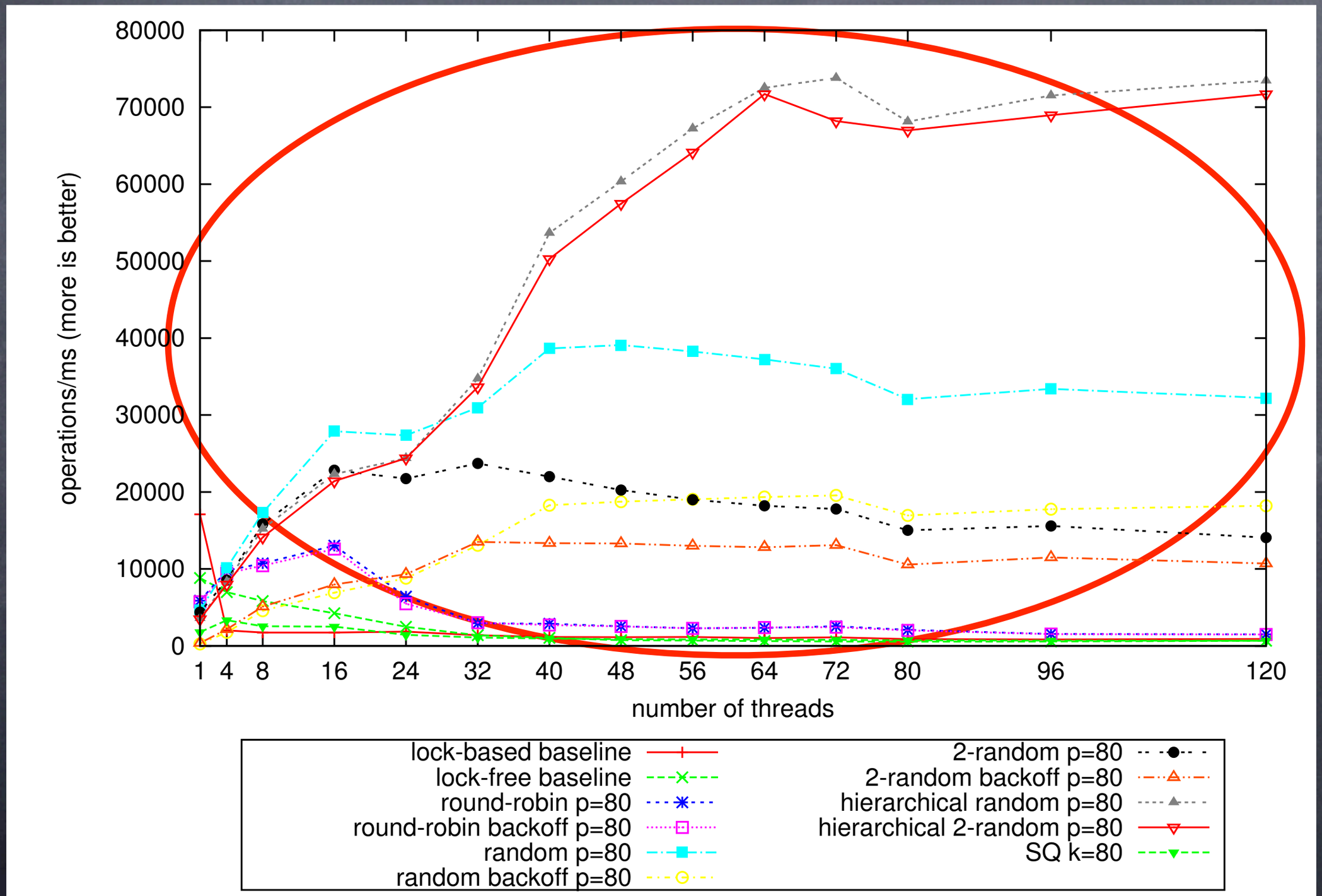
Ideal 40-Core Performance



Regular FIFO Queues



Our "Scal" Queues



Lock-Based (LB)

Lock-Based (LB)

Michael-Scott (MS)

[MS96]

Lock-Based (LB)

Flat Combining (FC)

[IST10]

Michael-Scott (MS)

[MS96]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Random Dequeue (RD)
[AKY10]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Lock-Based (LB)

Flat Combining (FC)

[IST10]

Michael-Scott (MS)

[MS96]

Random Dequeue (RD)

[AKY10]

Segment Queue (SQ)

[AKY10]

Round-Robin (RR)

[-PRS10]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Round-Robin (RR)
[-PRS10]

Random (RA)
[-PRS10]

Lock-Based (LB)

Flat Combining (FC)

[IST10]

Michael-Scott (MS)

[MS96]

Random Dequeue (RD)

[AKY10]

Segment Queue (SQ)

[AKY10]

Round-Robin (RR)

[-PRS10]

Random (RA)

[-PRS10]

d-Random (dRA)

[-PRS10]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Regular FIFO

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Round-Robin (RR)
[-PRS10]

Random (RA)
[-PRS10]

d-Random (dRA)
[-PRS10]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Regular FIFO

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Workload-independent k-FIFO

Round-Robin (RR)
[-PRS10]

Random (RA)
[-PRS10]

d-Random (dRA)
[-PRS10]

Lock-Based (LB)

Michael-Scott (MS)
[MS96]

Flat Combining (FC)
[IST10]

Regular FIFO

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Workload-independent k-FIFO

Round-Robin (RR)
[-PRS10]

Workload-dependent k-FIFO

Random (RA)
[-PRS10]

d-Random (dRA)
[-PRS10]

Lock-Based (LB)

Flat Combining (FC)
[IST10]

Michael-Scott (MS)
[MS96]

Regular FIFO

Random Dequeue (RD)
[AKY10]

Segment Queue (SQ)
[AKY10]

Workload-independent k-FIFO

Round-Robin (RR)
[-PRS10]

Workload-dependent k-FIFO

Random (RA)
[-PRS10]

Probabilistic k-FIFO

d-Random (dRA)
[-PRS10]

k-FIFO Queues

- with a k-FIFO queue elements may be returned out-of-FIFO order up to k

k-FIFO Queues

- with a k-FIFO queue elements may be returned out-of-FIFO order up to k
- the oldest element is returned after at most k+1 dequeue operations that may return elements not younger than k (or return nothing)

k-FIFO Queues

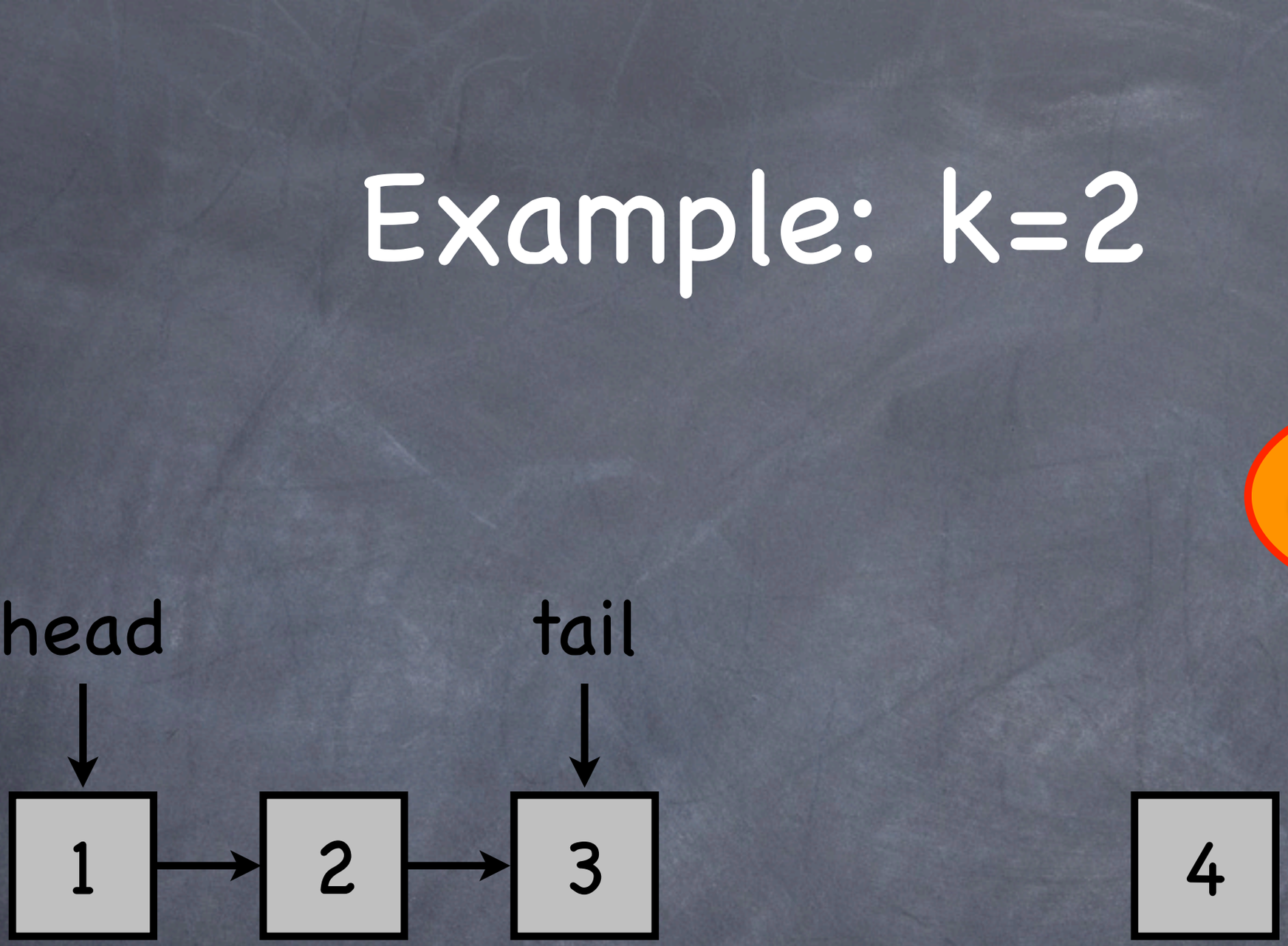
- with a k-FIFO queue elements may be returned out-of-FIFO order up to k
- the oldest element is returned after at most k+1 dequeue operations that may return elements not younger than k (or return nothing)
- starvation-free for finite k

k-FIFO Queues

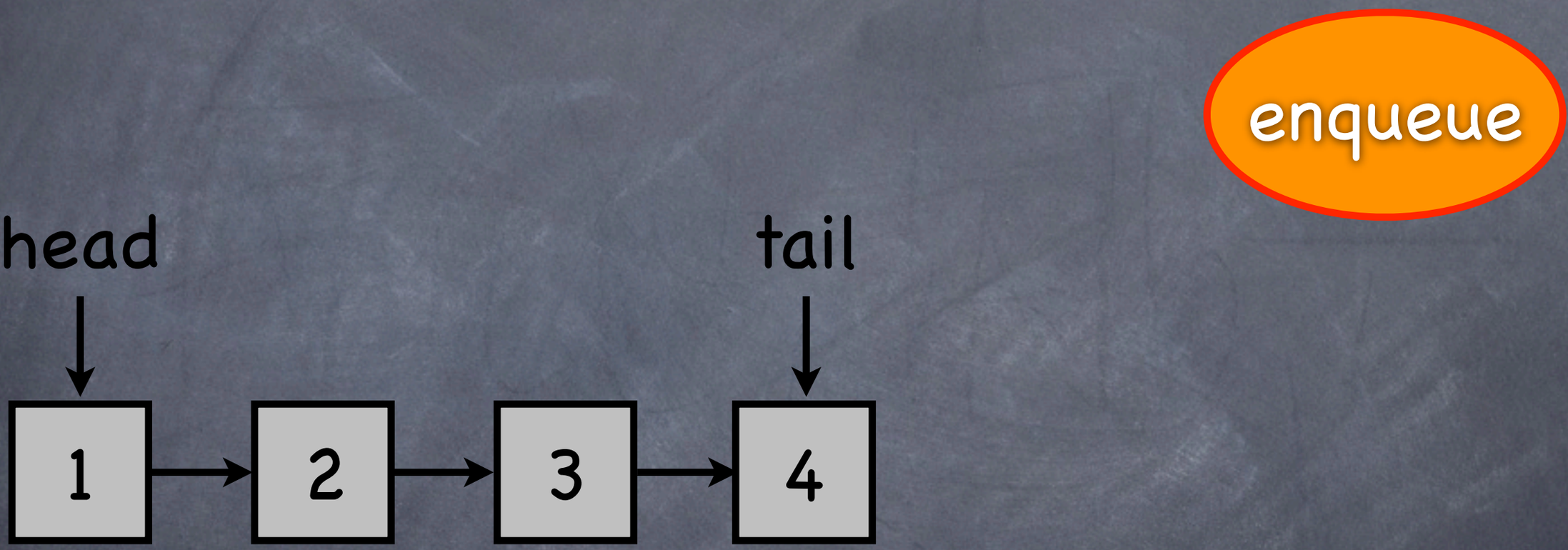
- with a k-FIFO queue elements may be returned out-of-FIFO order up to k
- the oldest element is returned after at most k+1 dequeue operations that may return elements not younger than k (or return nothing)
- starvation-free for finite k
- 0-FIFO queue = regular FIFO queue

Example: k=2

enqueue



Example: k=2



Example: k=2

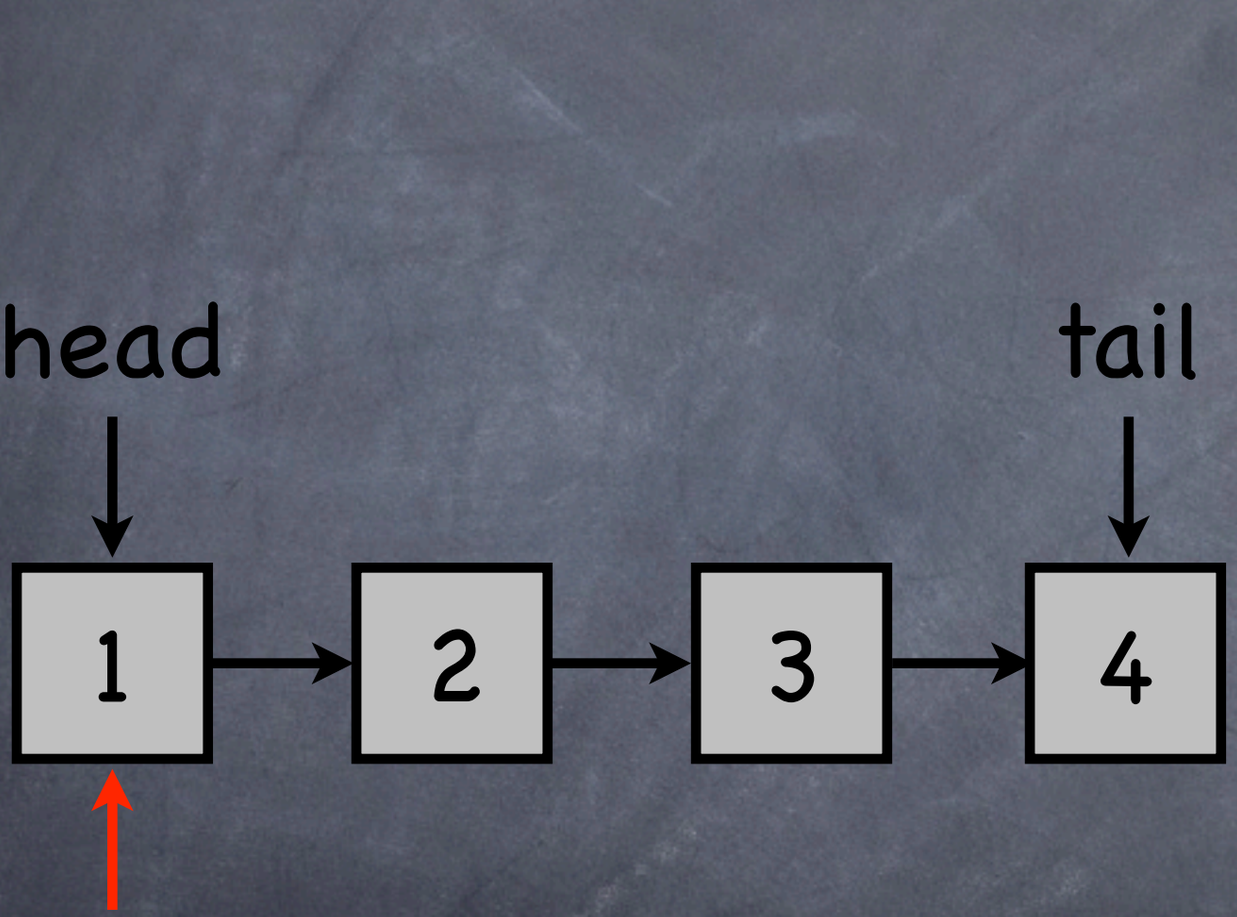
dequeue

head

tail

Example: k=2

dequeue

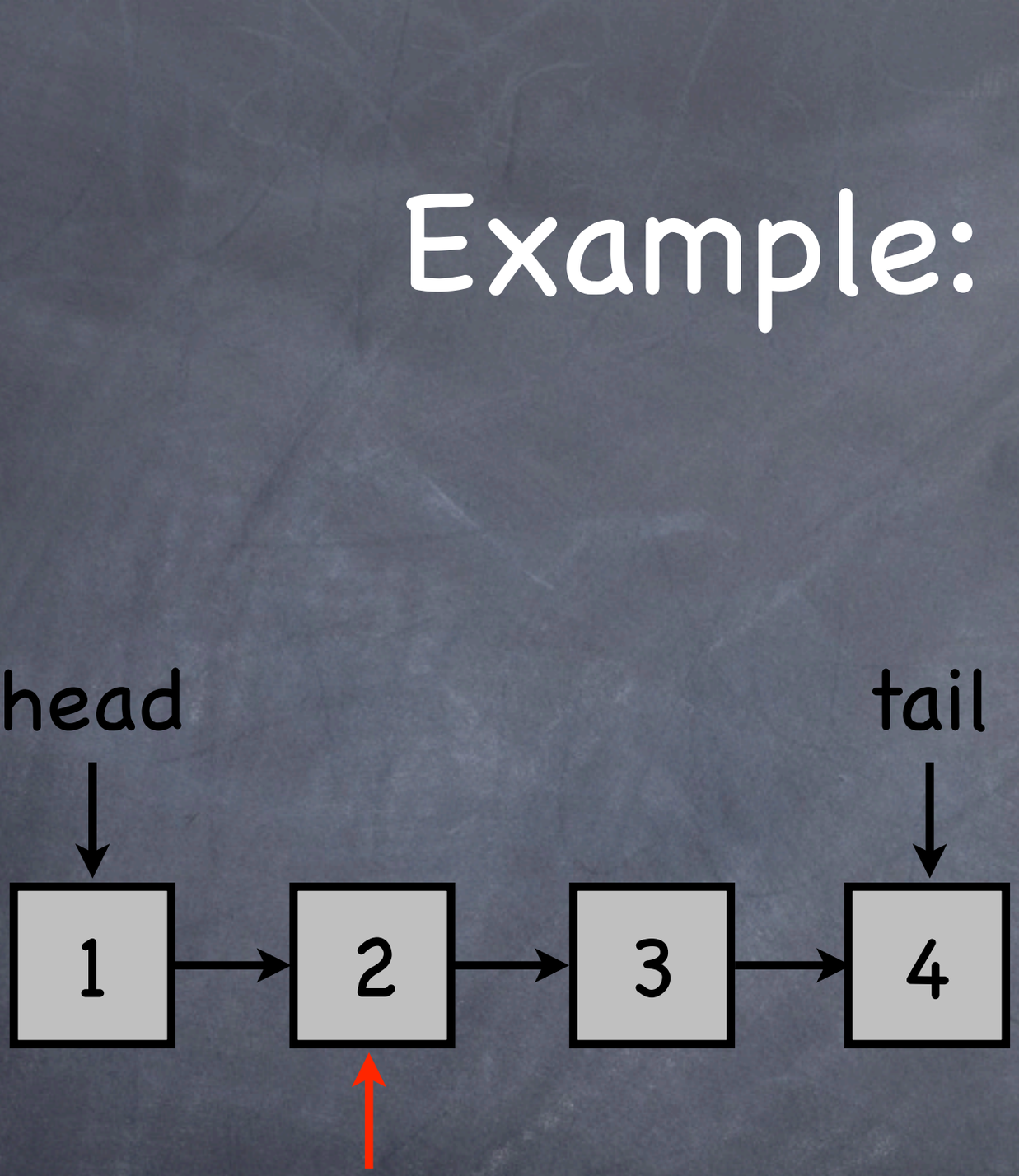


Example: k=2

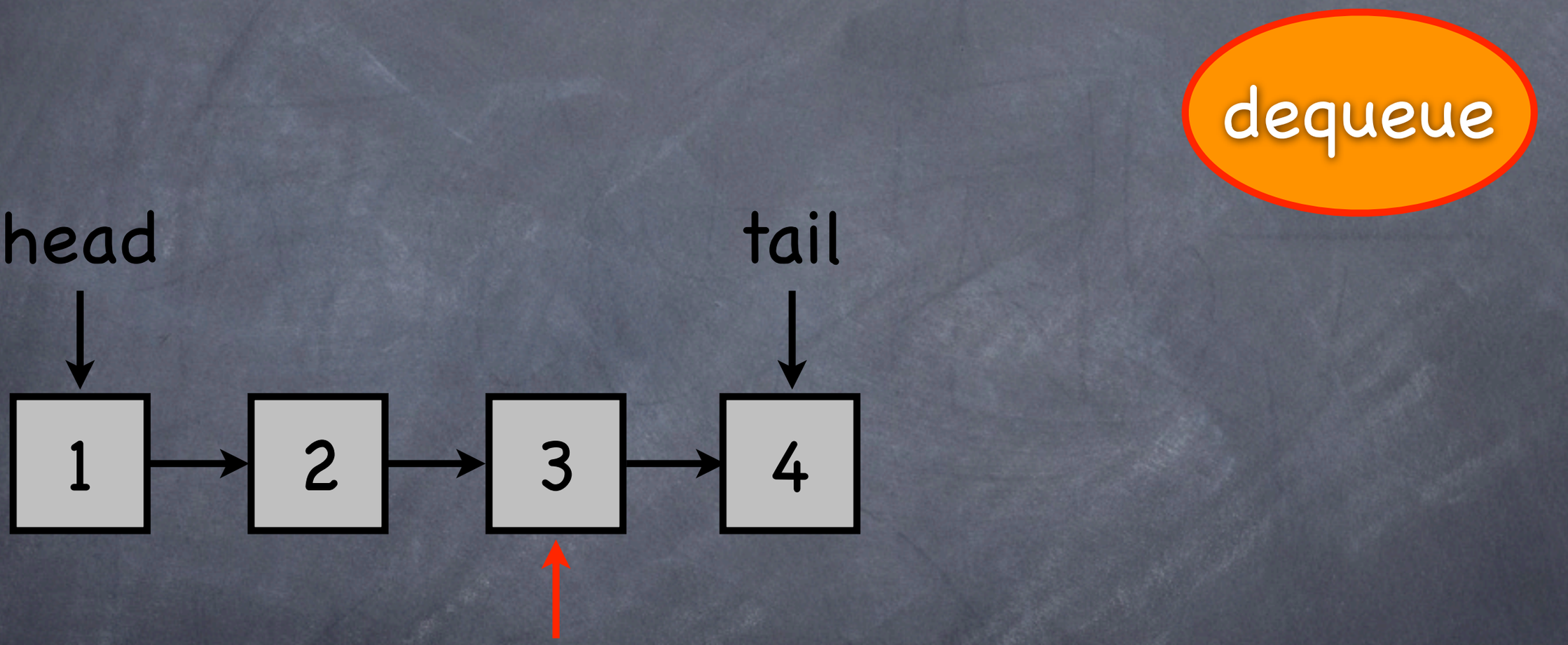
dequeue

head

tail



Example: k=2

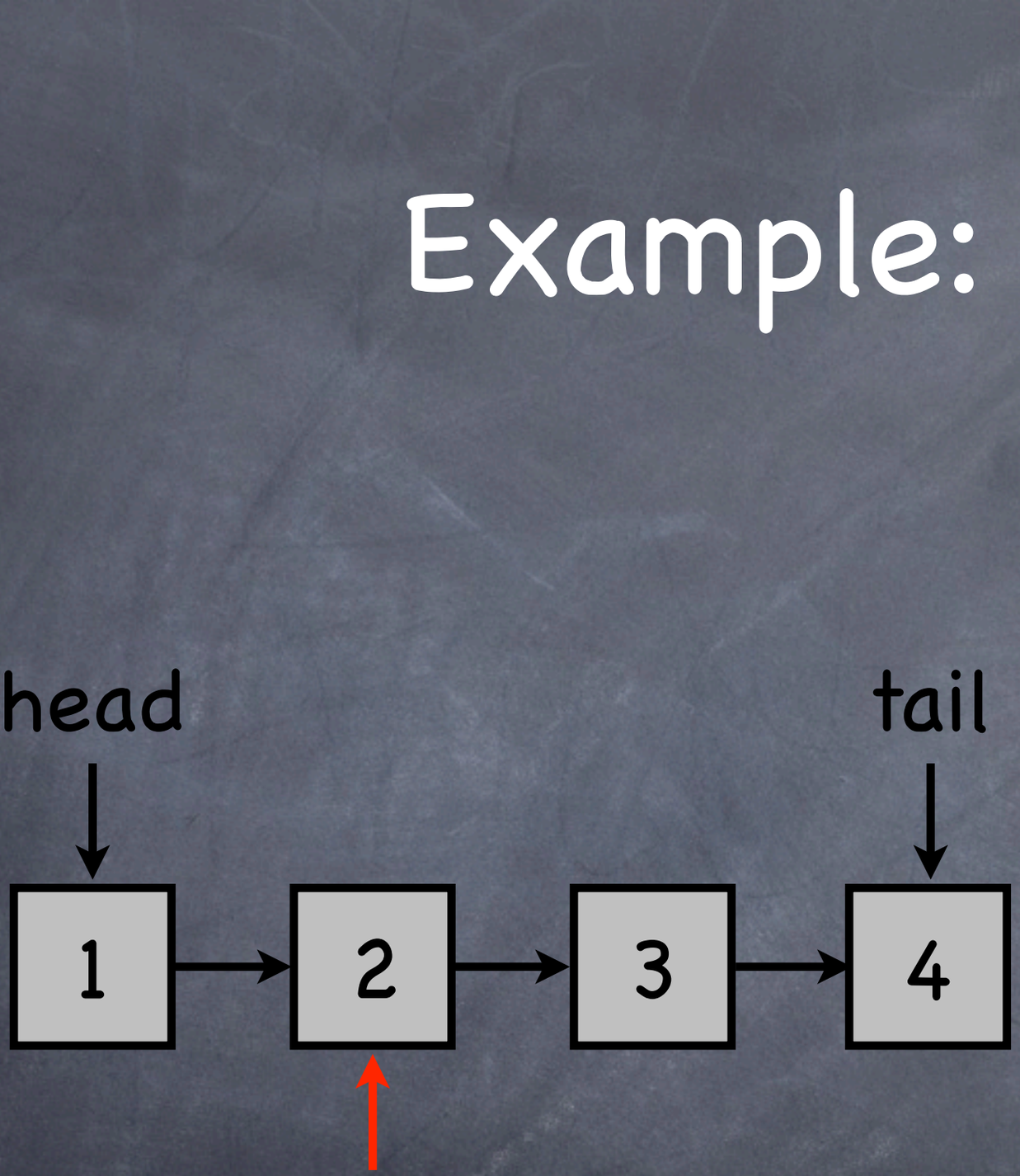


Example: k=2

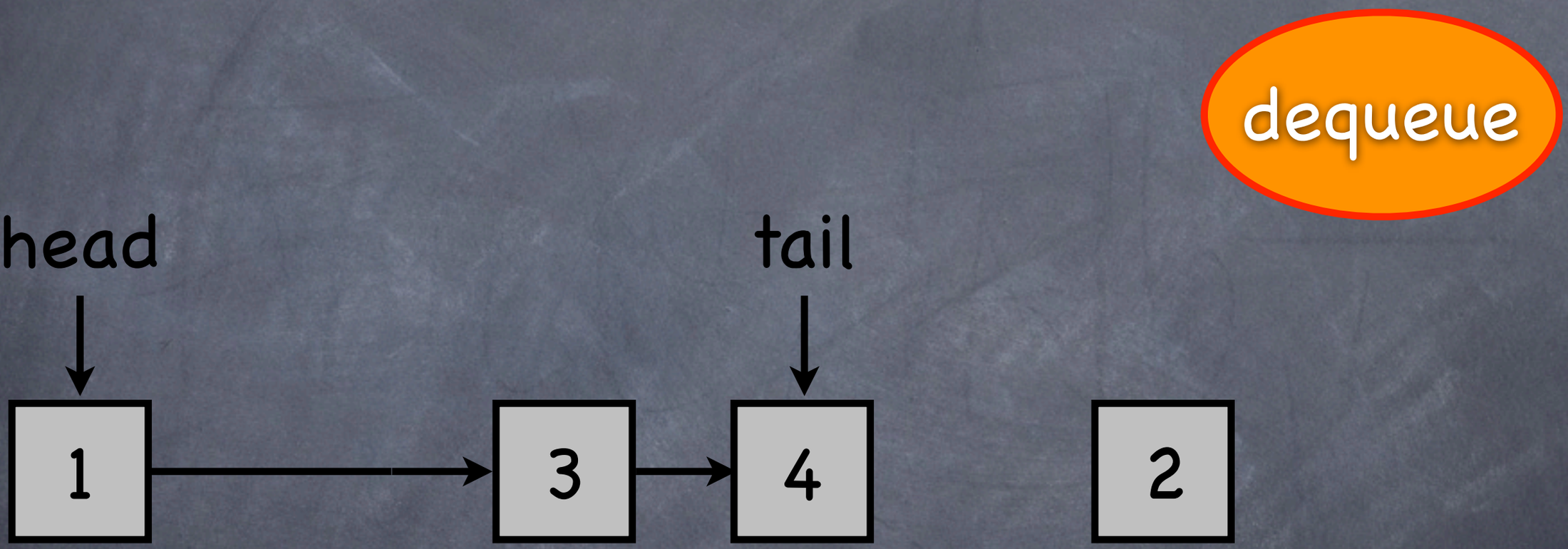
dequeue

head

tail

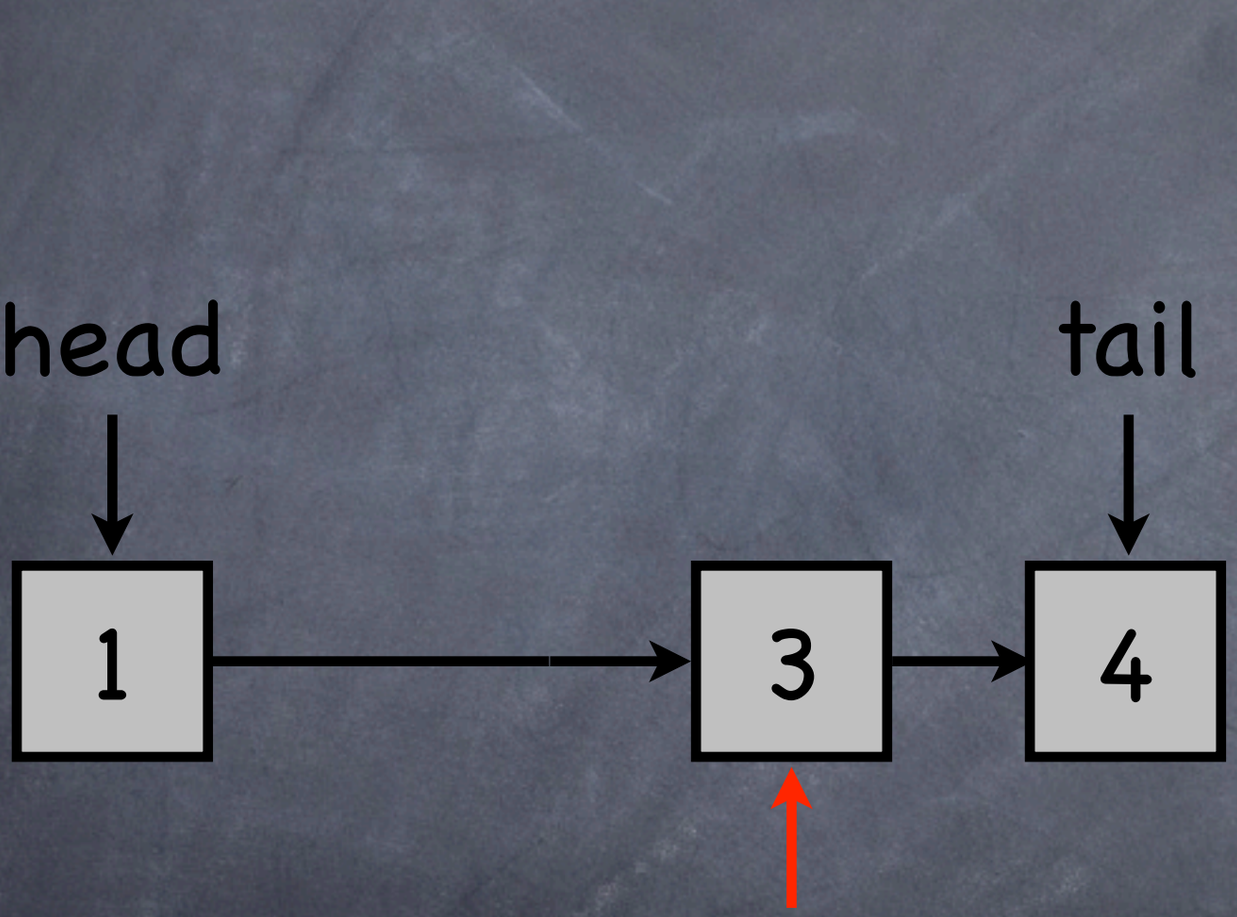


Example: k=2

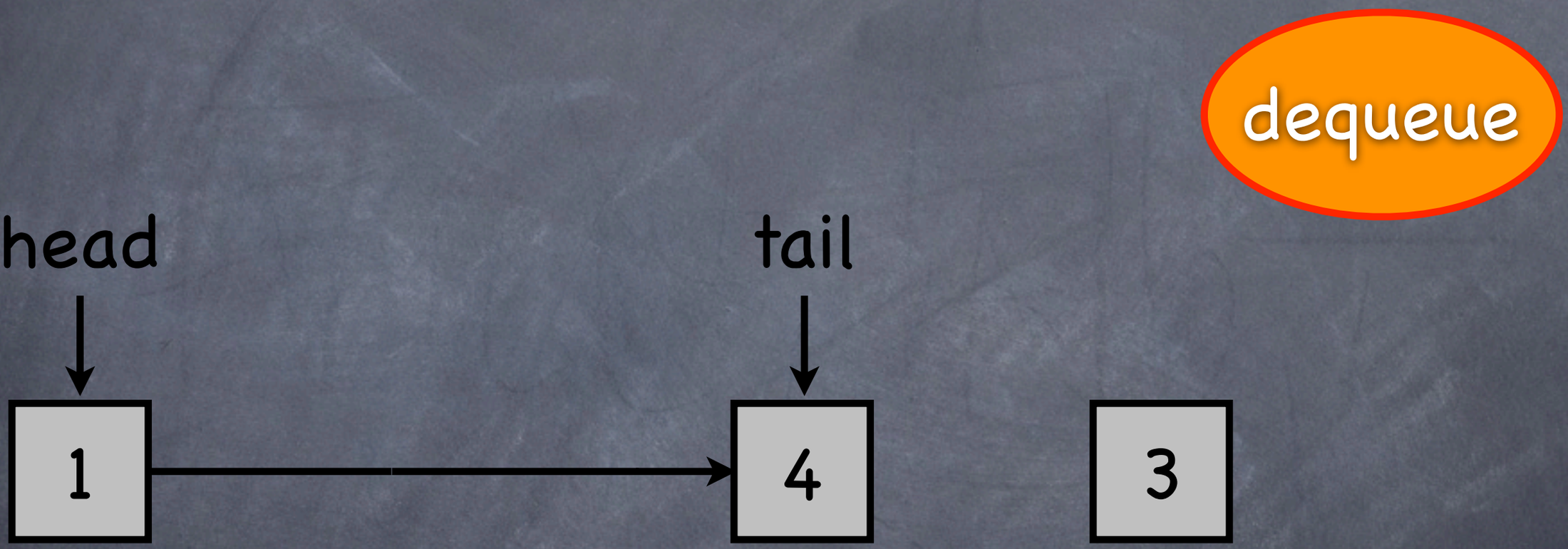


Example: k=2

dequeue



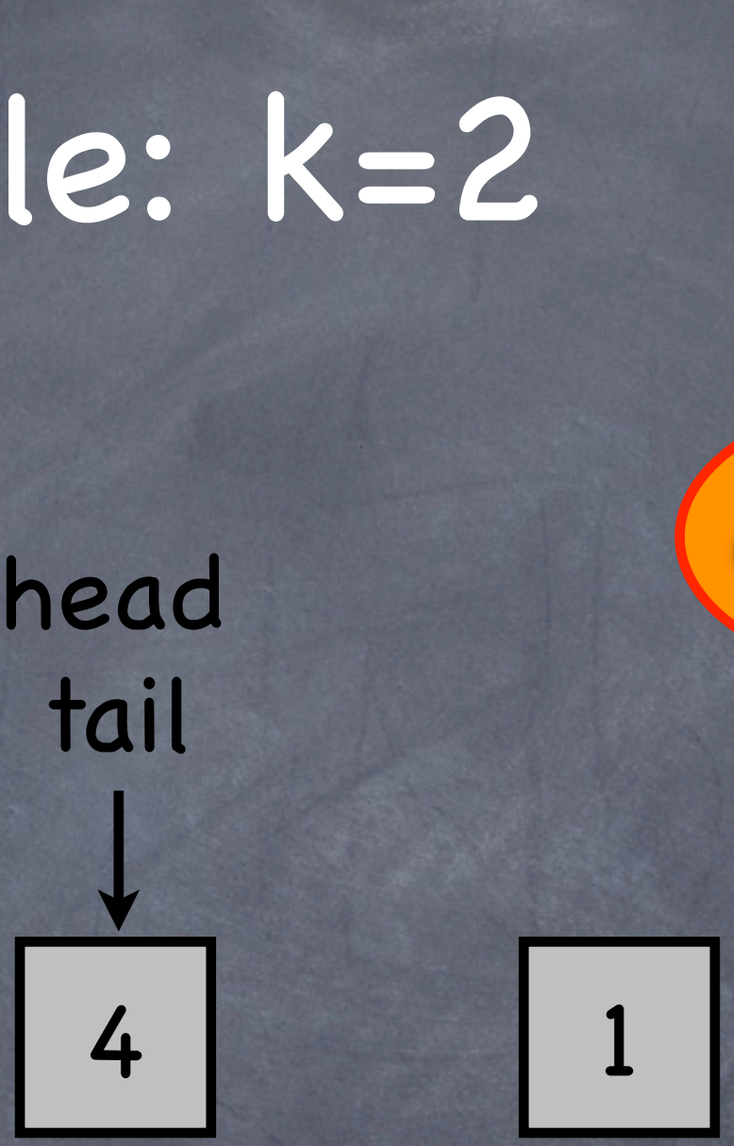
Example: k=2



Example: k=2

dequeue

head
tail



Worst-case Semantical Deviation (WCSD)

- we call k the worst-case semantical deviation (WCSD) of a k-FIFO queue from a regular FIFO queue

Worst-case Semantical Deviation (WCSD)

- we call k the worst-case semantical deviation (WCSD) of a k-FIFO queue from a regular FIFO queue
- k may be zero, i.e., there is no semantical deviation (LB, MS, FC)

Worst-case Semantical Deviation (WCSD)

- we call k the worst-case semantical deviation (WCSD) of a k-FIFO queue from a regular FIFO queue

- k may be zero, i.e., there is no semantical deviation (LB, MS, FC)

- k may be configurable and independent of any workload (RD, SQ)

Worst-case Semantical Deviation (WCSD)

- we call k the worst-case semantical deviation (WCSD) of a k-FIFO queue from a regular FIFO queue

- k may be zero, i.e., there is no semantical deviation (LB, MS, FC)

- k may be configurable and independent of any workload (RD, SQ)

- k may also be workload-dependent (RR) and even probabilistic (RA, dRA)

WCSD of existing k-FIFO Queue Implementations

Queue Implementation	k	o
Lock-Based (LB)	0	0
Lock-free Michael-Scott (MS) [1]	0	0
Flat Combining (FC) [2]	0	0
Random Dequeue Queue (RD) [3]	r	0
Segment Queue (SQ) [3]	s	∞

[1] M. Michael and M. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In Proc. PODC, pages 267-275. ACM, 1996.
[2] D.H.I. Incze, N. Shavit, and M. Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In Proc. SPAA, pages 355-364. ACM, 2010
[3] Y. Afek, G. Korland, and E. Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In Proc. OPODIS, pages 395-410. Springer, 2010.

WCSD of existing k-FIFO Queue Implementations

Queue Implementation	k	o
Lock-Based (LB)	0	0
Lock-free Michael-Scott (MS) [1]	0	0
Flat Combining (FC) [2]	0	0
Random D	r	0
Segme	s	∞

regular FIFO queues

[1] M. Michael and M. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In Proc. PODC, pages 267-275. ACM, 1996.
[2] D.H.I. Incze, N. Shavit, and M. Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In Proc. SPAA, pages 355-364. ACM, 2010
[3] Y. Afek, G. Korland, and E. Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In Proc. OPODIS, pages 395-410. Springer, 2010.

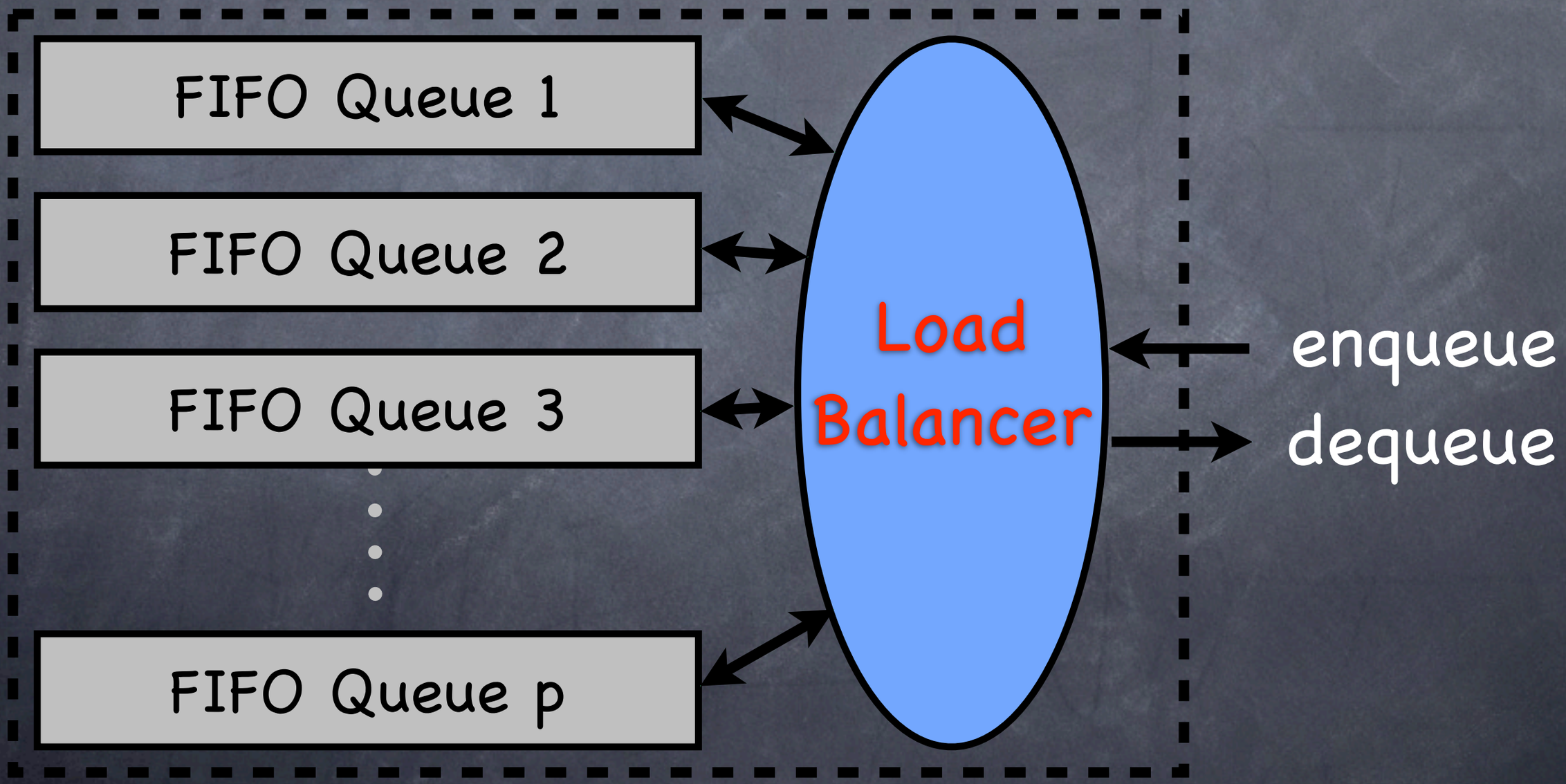
WCSD of existing k-FIFO Queue Implementations

Queue Implementation	k	o
Lock	0	0
Lock	0	0
Flat	0	0
Random Dequeue Queue (RD) [3]	r	0
Segment Queue (SQ) [3]	s	∞

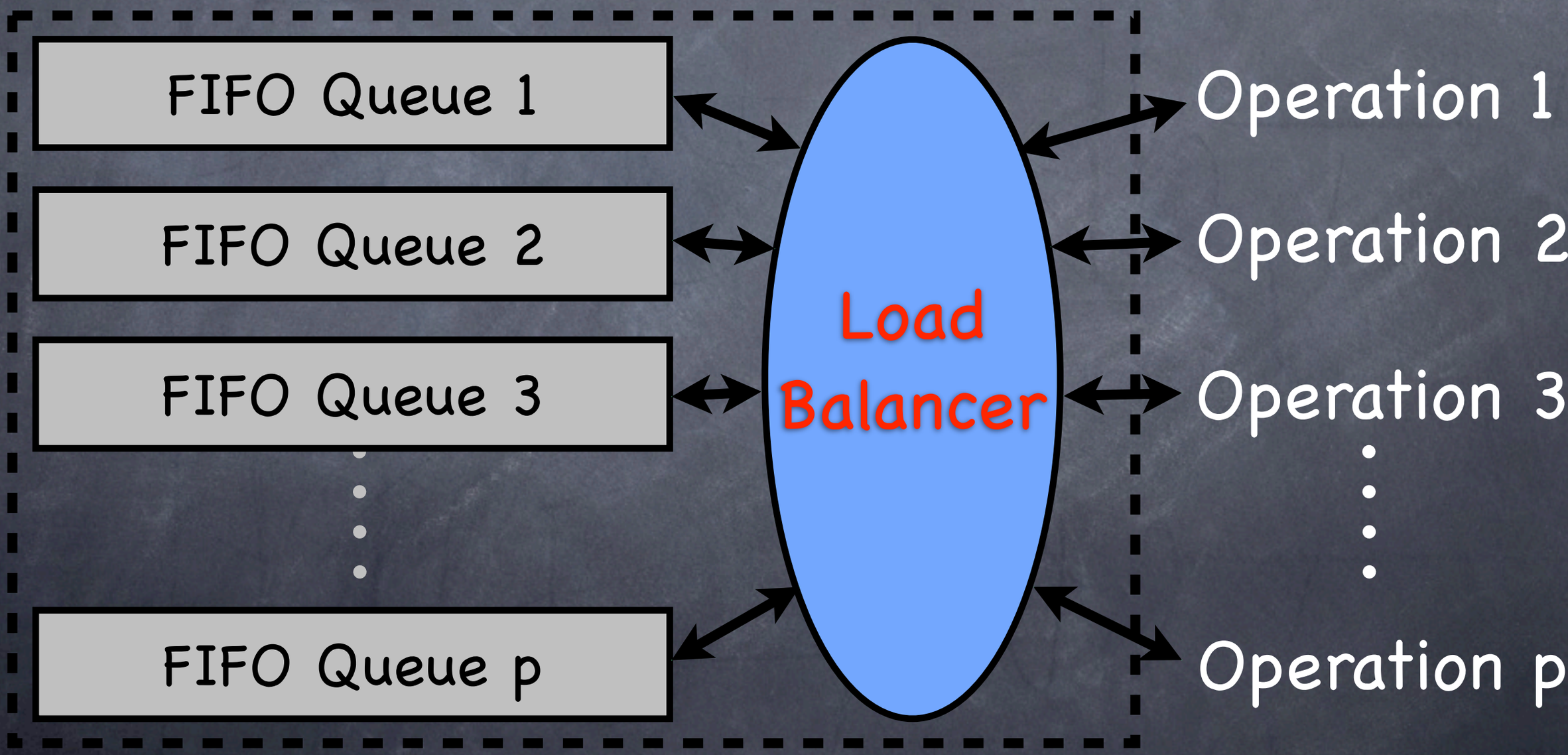
workload-independent constant bounds

[1] M. Michael and M. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In Proc. PODC, pages 267-275. ACM, 1996.
[2] D.H.I. Incze, N. Shavit, and M. Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In Proc. SPAA, pages 355-364. ACM, 2010
[3] Y. Afek, G. Korland, and E. Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In Proc. OPODIS, pages 395-410. Springer, 2010.

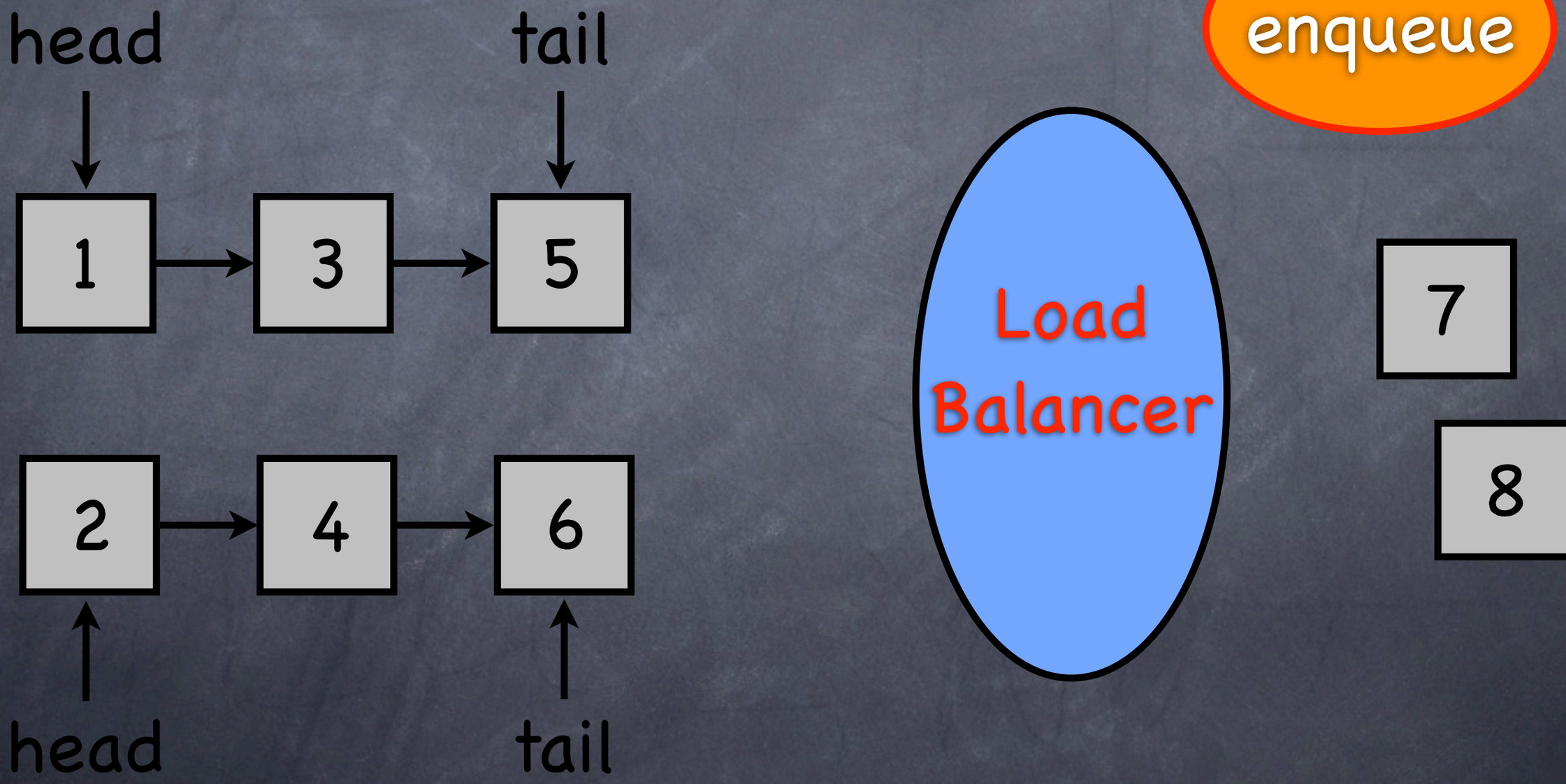
Scal Queue:
p FIFO Queues



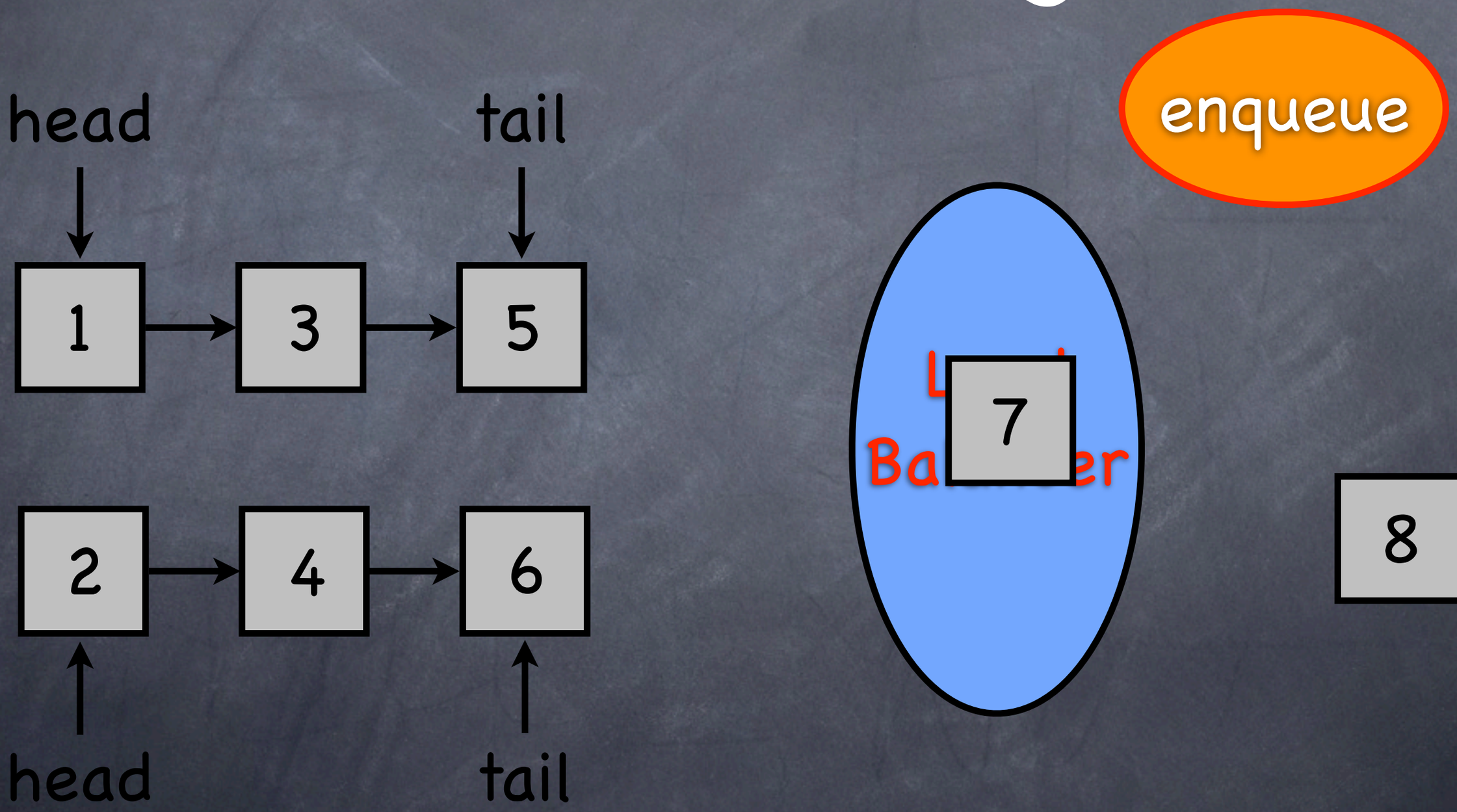
Scal Queue:
Up to p Parallel Operations



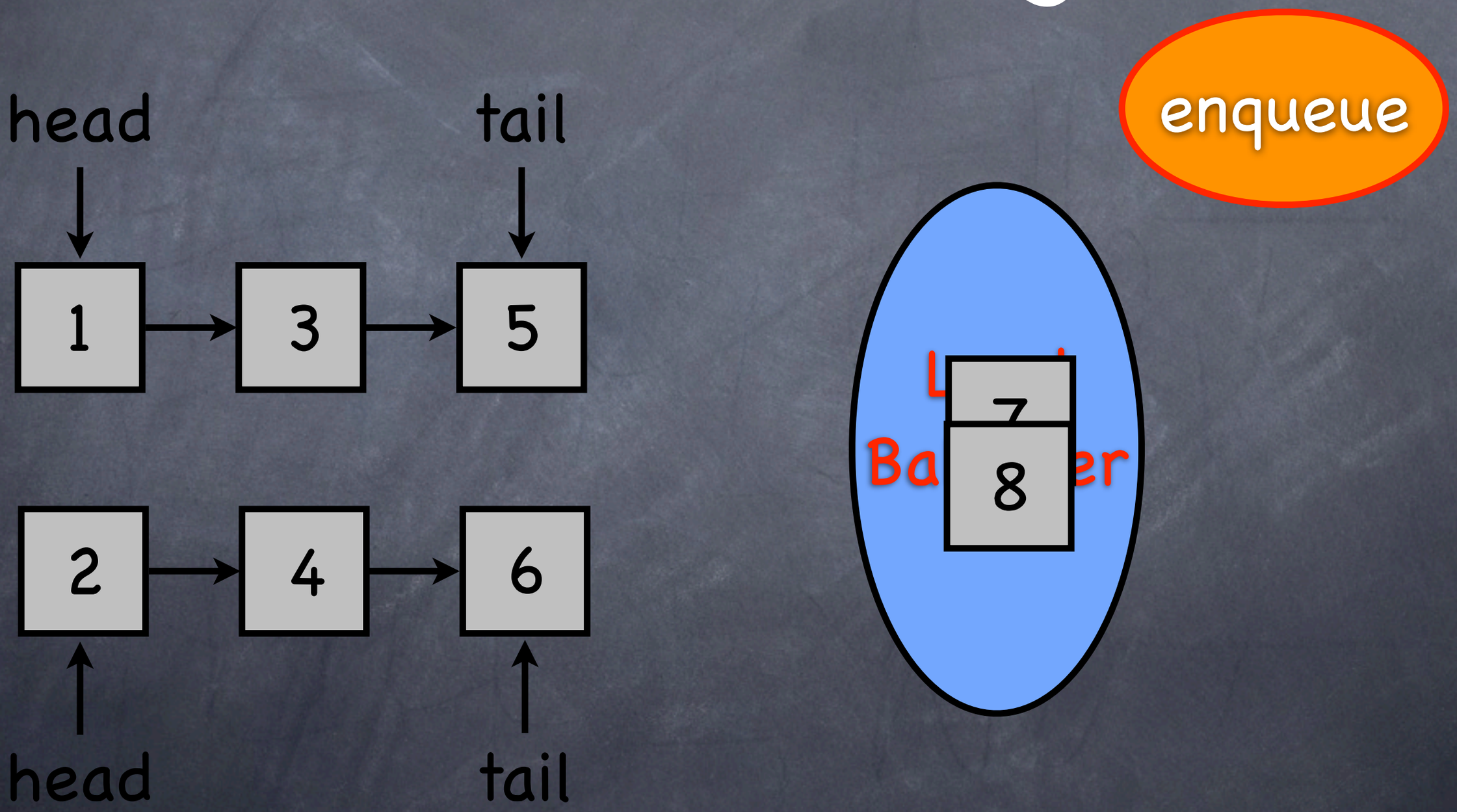
Scal Queue: Load Balancing



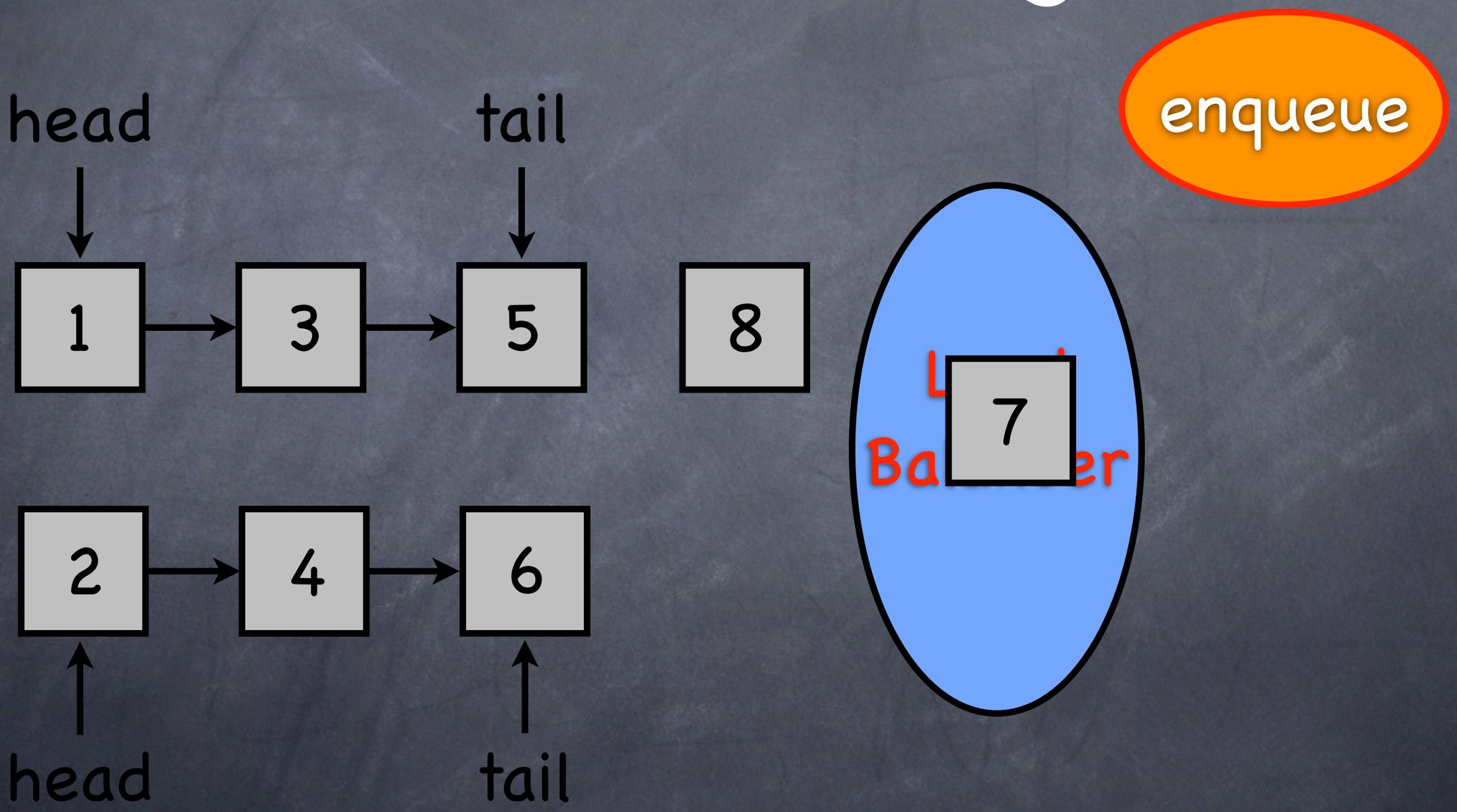
Scal Queue: Load Balancing



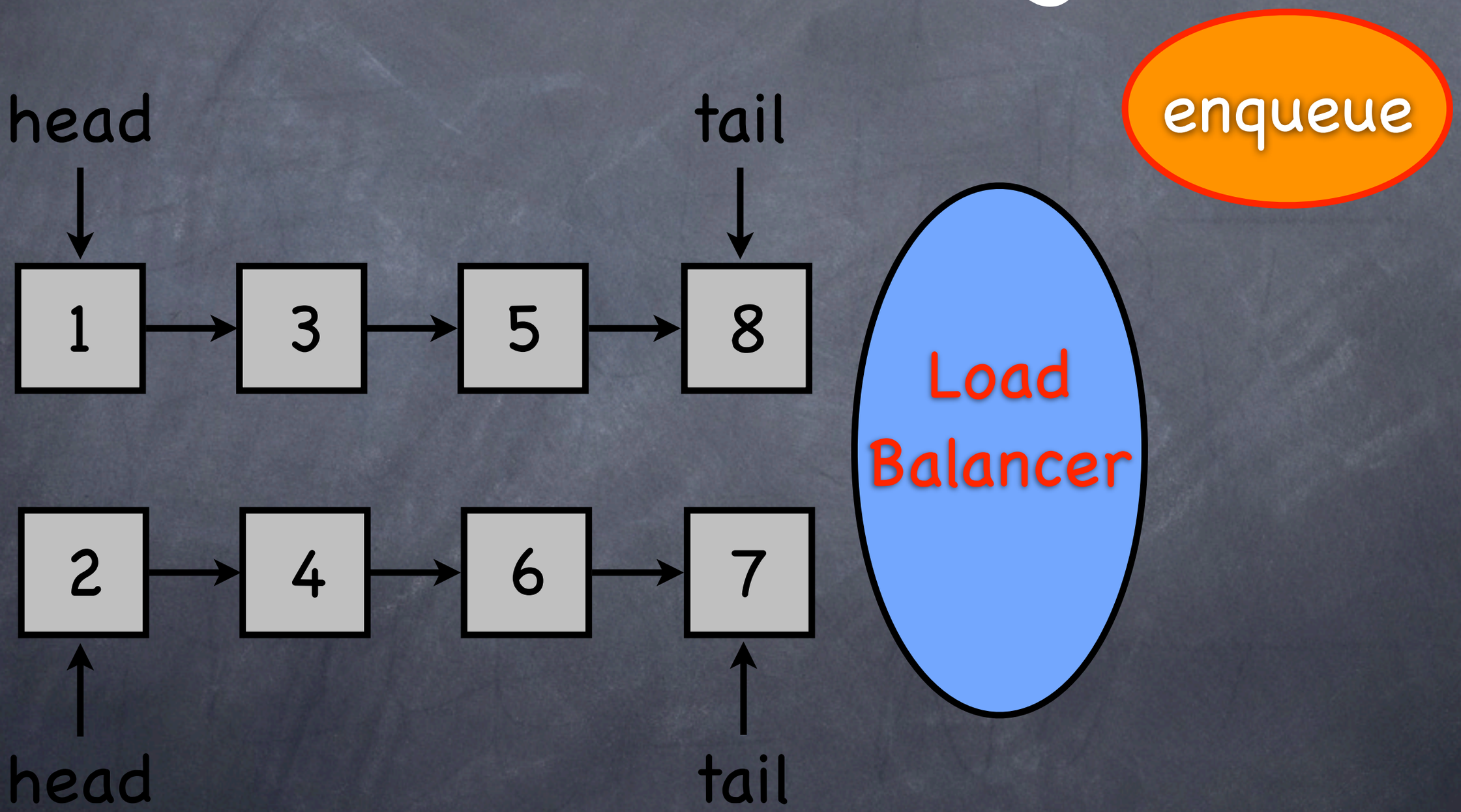
Scal Queue: Load Balancing



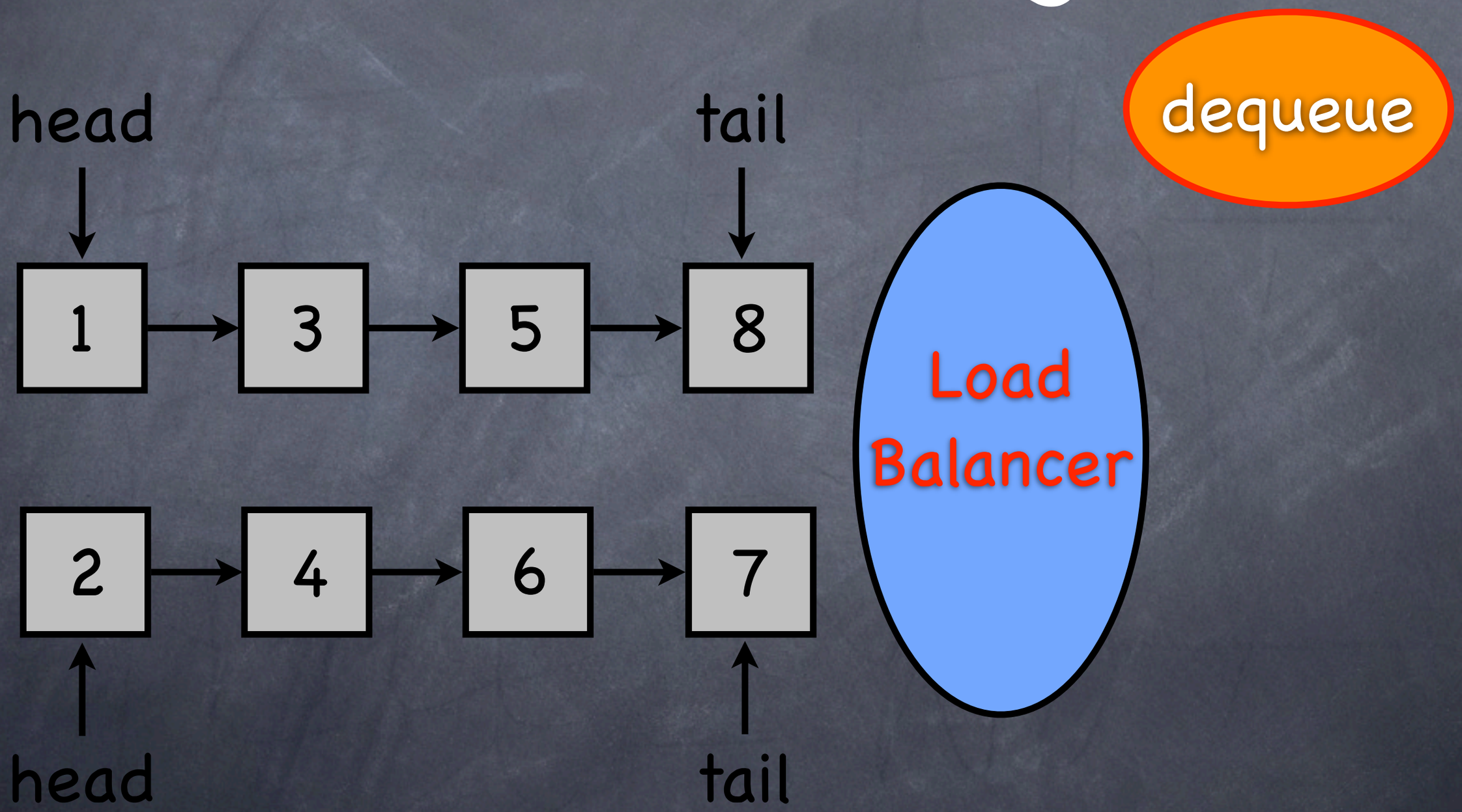
Scal Queue: Load Balancing



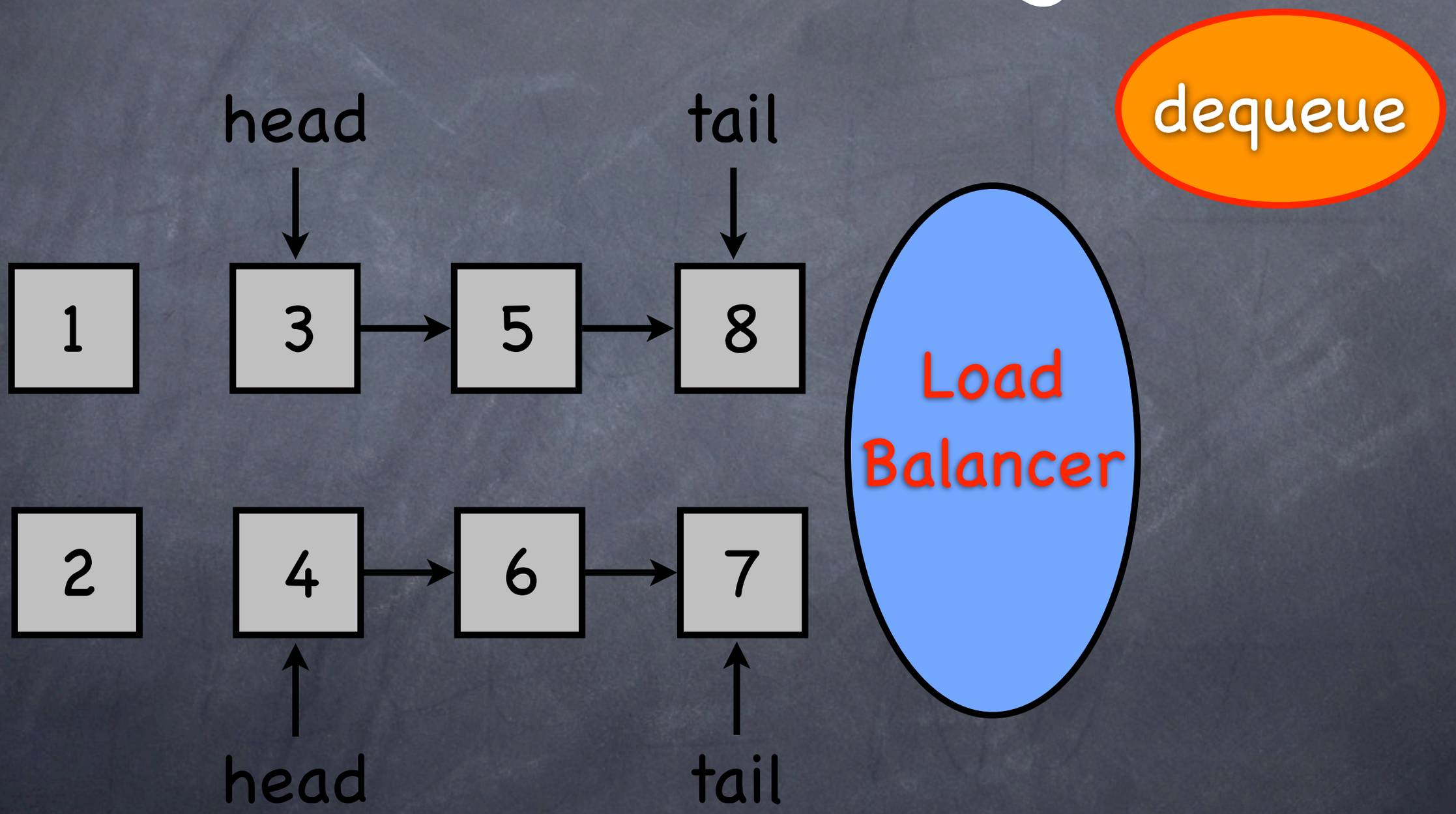
Scal Queue: Load Balancing



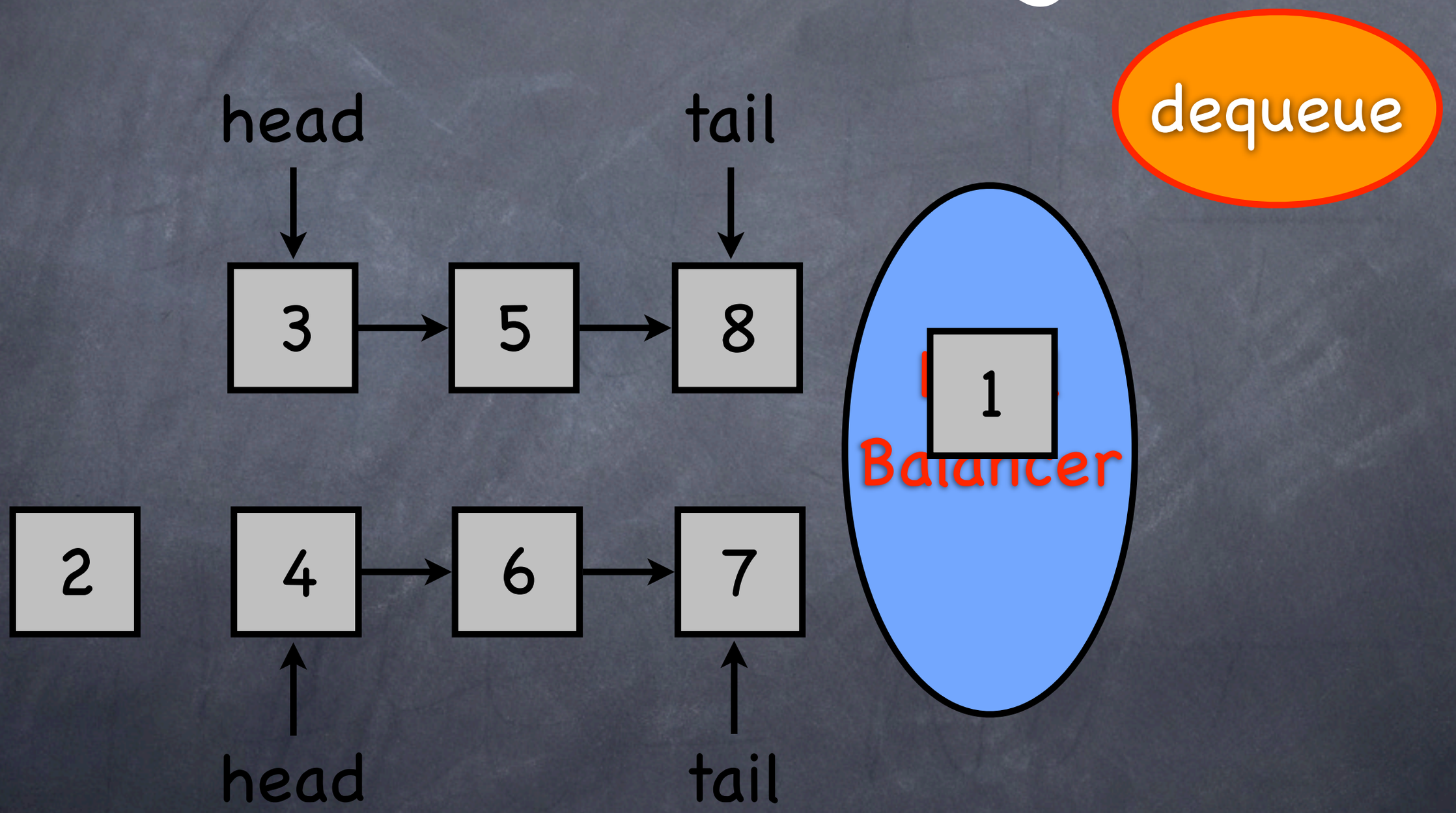
Scal Queue: Load Balancing



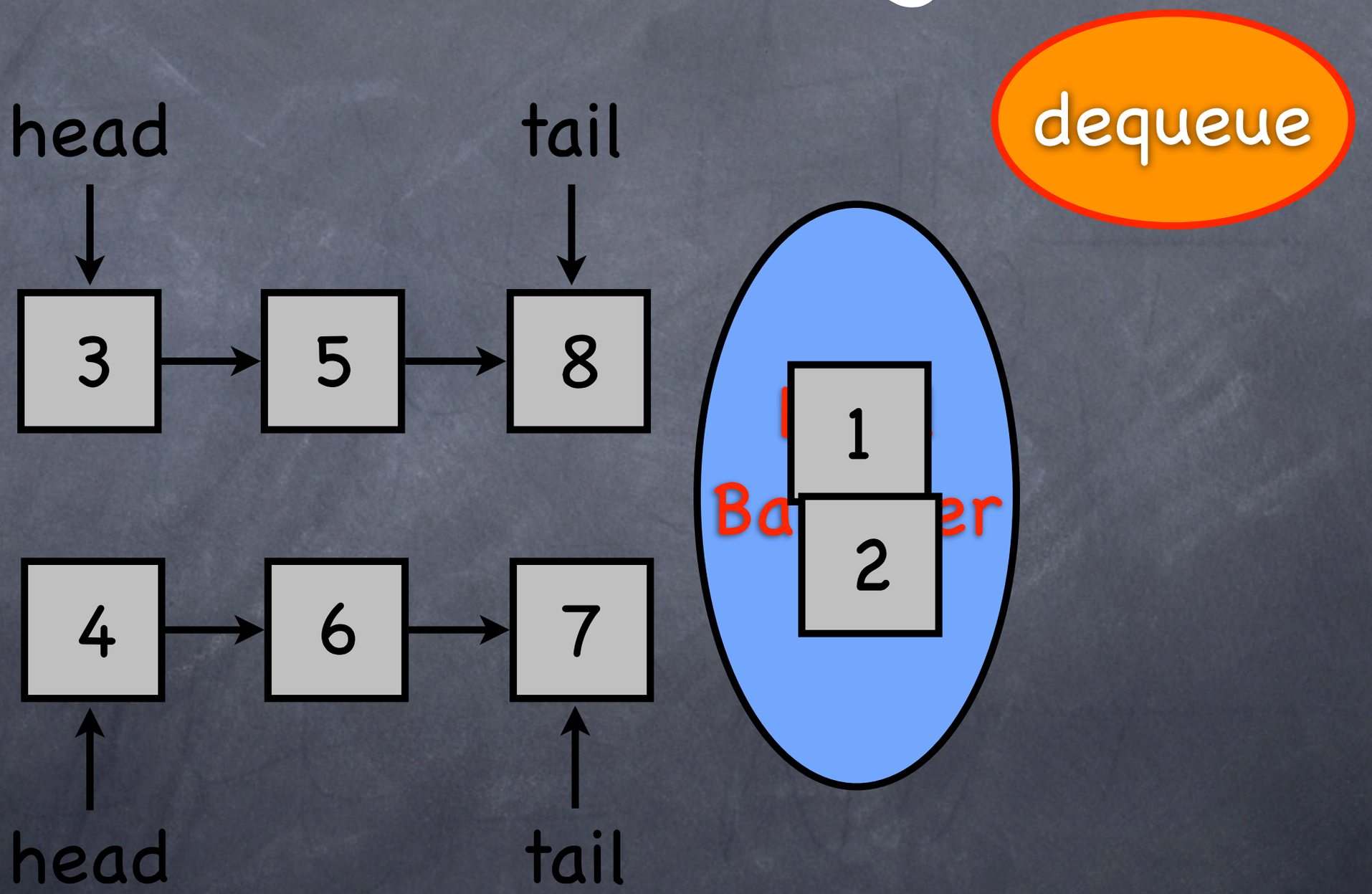
Scal Queue: Load Balancing



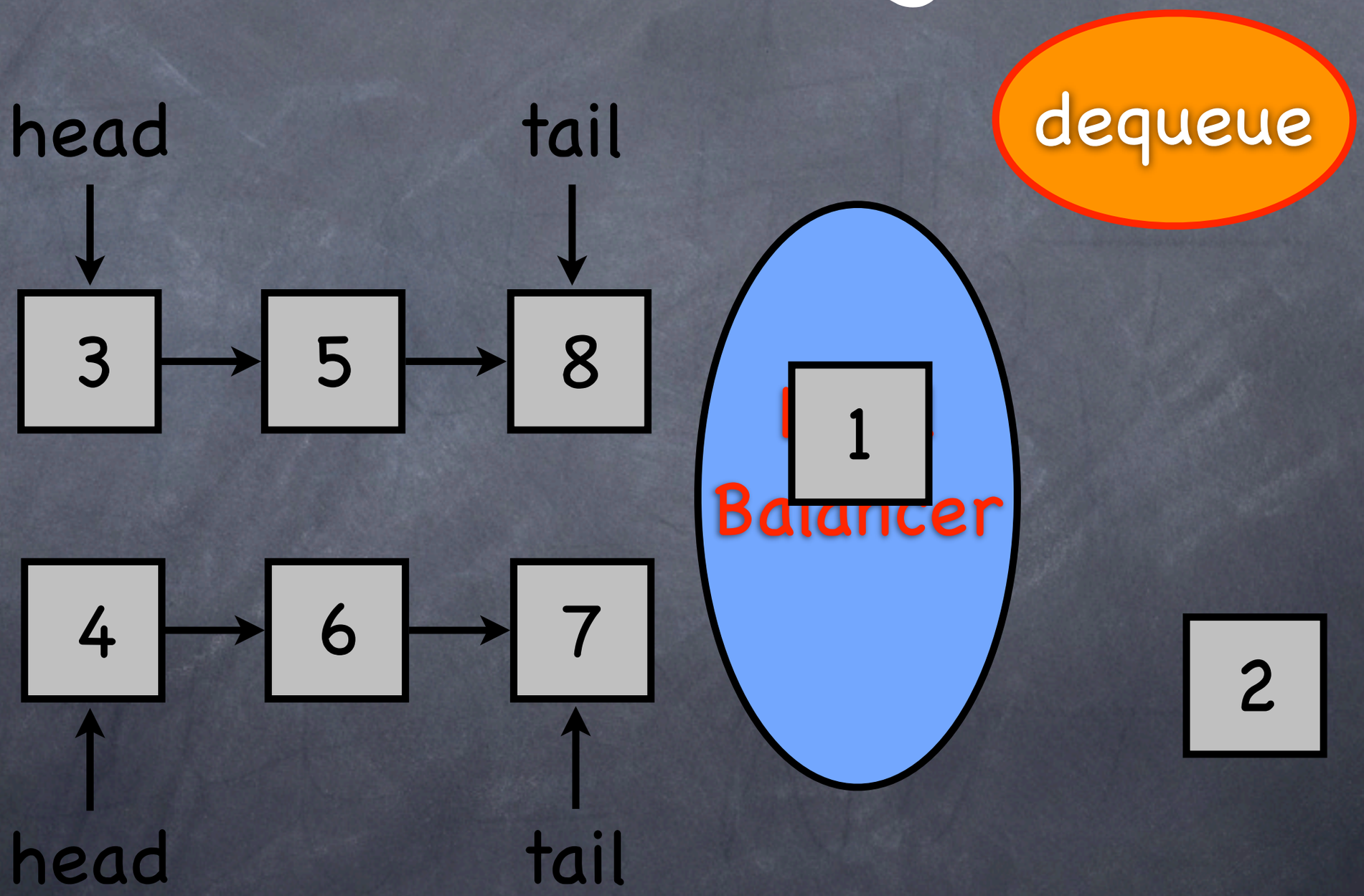
Scal Queue: Load Balancing



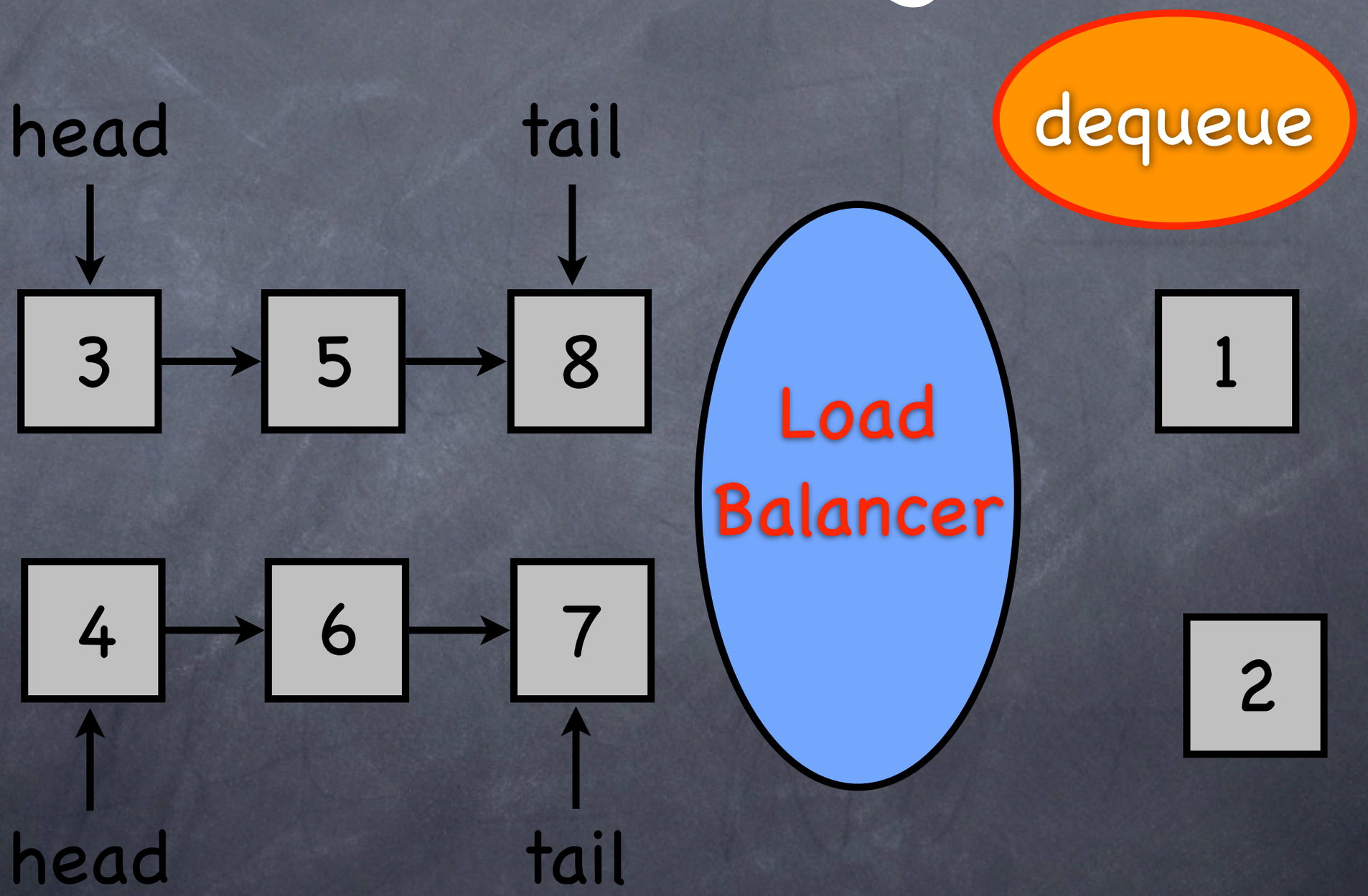
Scal Queue: Load Balancing



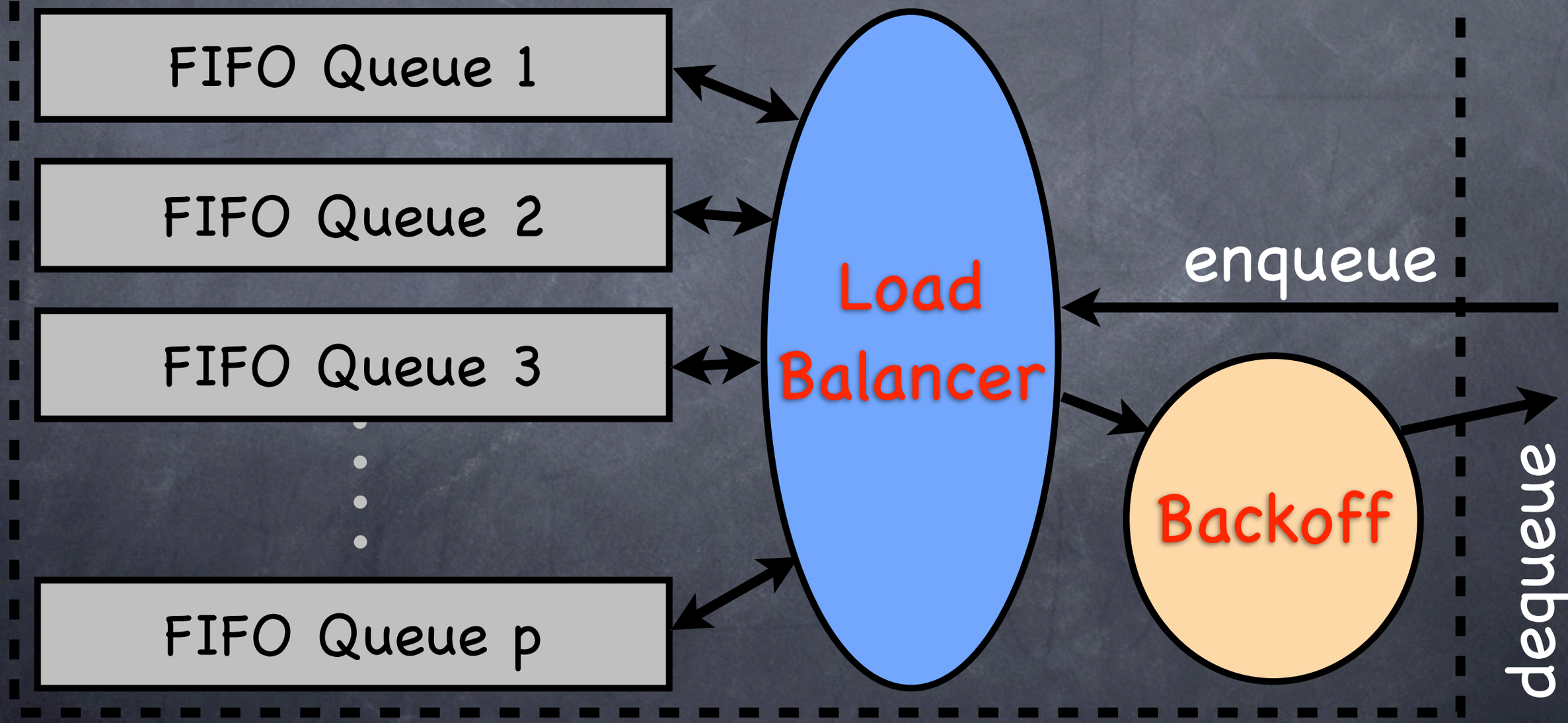
Scal Queue: Load Balancing



Scal Queue: Load Balancing



Scal Queue: Backoff



WCSD of Scal Queues

Load balancer	k	o
Round-Robin (RR)	$t \cdot (p-1)$	$t \cdot (p-1)$
Round-Robin Backoff (RR-B)	$t \cdot (p-1)$	0
Random (RA)	$2 \cdot R \cdot (p-1)$	$2 \cdot R \cdot (p-1)$
Random Backoff (RA-B)	$2 \cdot R \cdot (p-1)$	0
2-Random (2RA)	$2 \cdot Q \cdot (p-1)$	$2 \cdot Q \cdot (p-1)$
2-Random Backoff (2RA-B)	$2 \cdot Q \cdot (p-1)$	0
Hierarchical 2-Random (H-2RA)	$2 \cdot Q \cdot (p-1)$	$2 \cdot Q \cdot (p-1)$
Hierarchical 2-Random Backoff (H-2RA-B)	$2 \cdot Q \cdot (p-1)$	0

$$t \text{ threads}, R = \Theta\left(\sqrt{\frac{t \cdot m \cdot \log p}{p}}\right), Q = \Theta\left(\frac{\log \log p}{d}\right)$$

WCSD of Scal Queues

Load balancer	k	o
Round-Robin (RR)	$t \cdot (p-1)$	$t \cdot (p-1)$
Round-Robin Backoff (RR-B)	$t \cdot (p-1)$	0
Random	$2 \cdot R \cdot (p-1)$	$2 \cdot R \cdot (p-1)$
Random	$(p-1)$	0
2-R	$1)$	$2 \cdot Q \cdot (p-1)$
2-Random	$-1)$	0
Hierarchical 2-R	$Q \cdot (p-1)$	$2 \cdot Q \cdot (p-1)$
Hierarchical 2-Random Backoff (H-2RA-B)	$2 \cdot Q \cdot (p-1)$	0

bounded in number of threads (t) and partial FIFO queues (p)

$$t \text{ threads}, R = \Theta\left(\sqrt{\frac{t \cdot m \cdot \log p}{p}}\right), Q = \Theta\left(\frac{\log \log p}{d}\right)$$

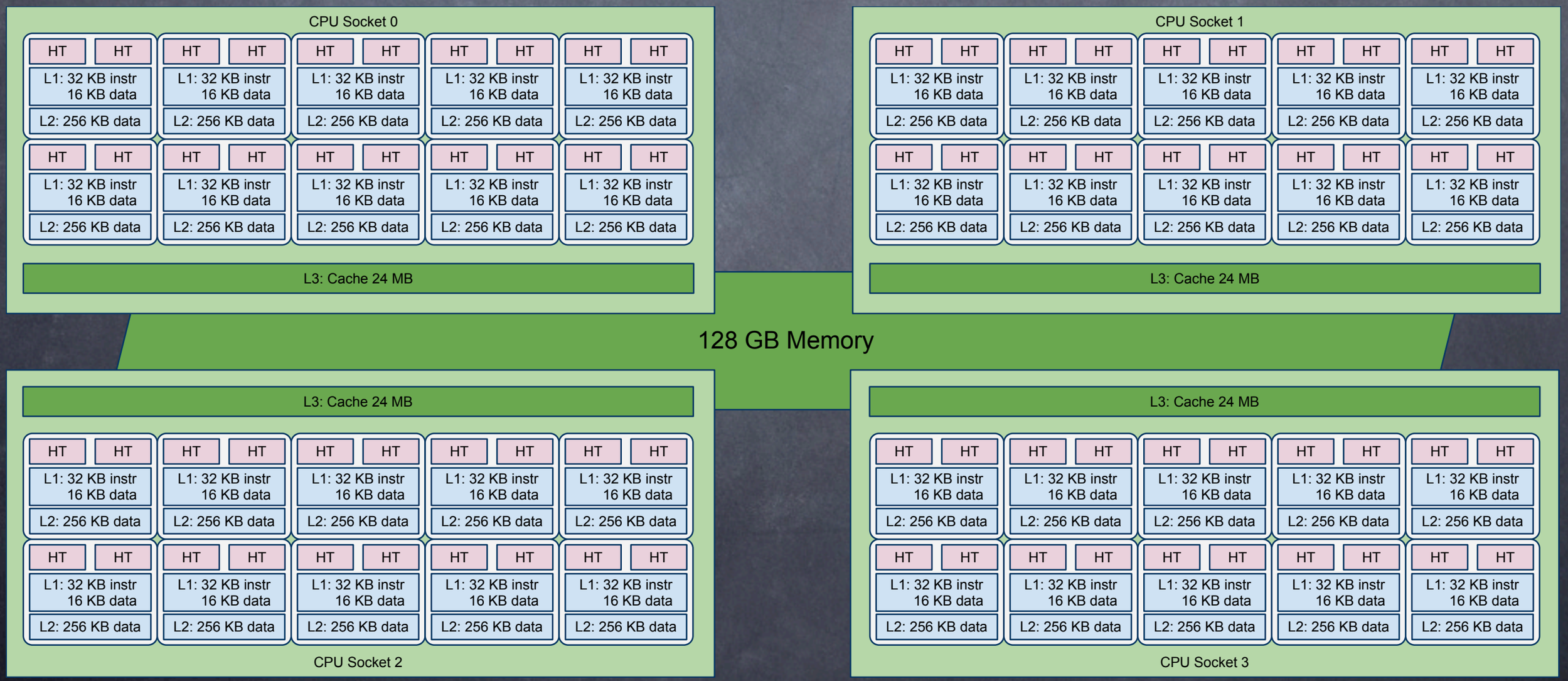
WCSD of Scal Queues

Load b	k	o
Roun		t · (p–1)
Round-Robin Ba	(p–1)	0
Random (RA)	2 · R · (p–1)	2 · R · (p–1)
Random Backoff (RA-B)	2 · R · (p–1)	0
2-Random (2RA)	2 · Q · (p–1)	2 · Q · (p–1)
2-Random Backoff (2RA-B)	2 · Q · (p–1)	0
Hierarchical 2-Random (H-2RA)	2 · Q · (p–1)	2 · Q · (p–1)
Hierarchical 2-Random Backoff (H-2RA-B)	2 · Q · (p–1)	0

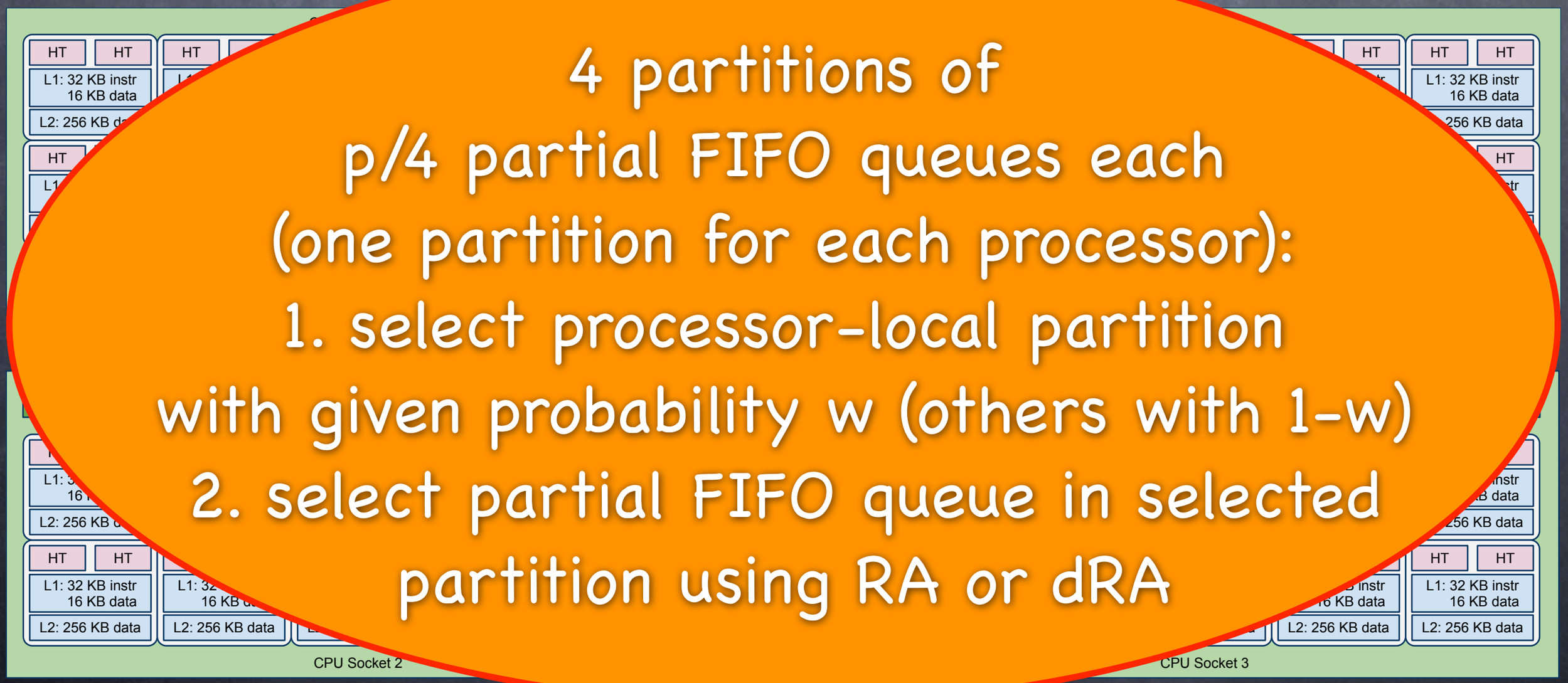
bounded probabilistically

$$t \text{ threads}, R = \Theta\left(\sqrt{\frac{t \cdot m \cdot \log p}{p}}\right), Q = \Theta\left(\frac{\log \log p}{d}\right)$$

4 processors x 10 cores x 2 hardware threads = 80 hardware threads



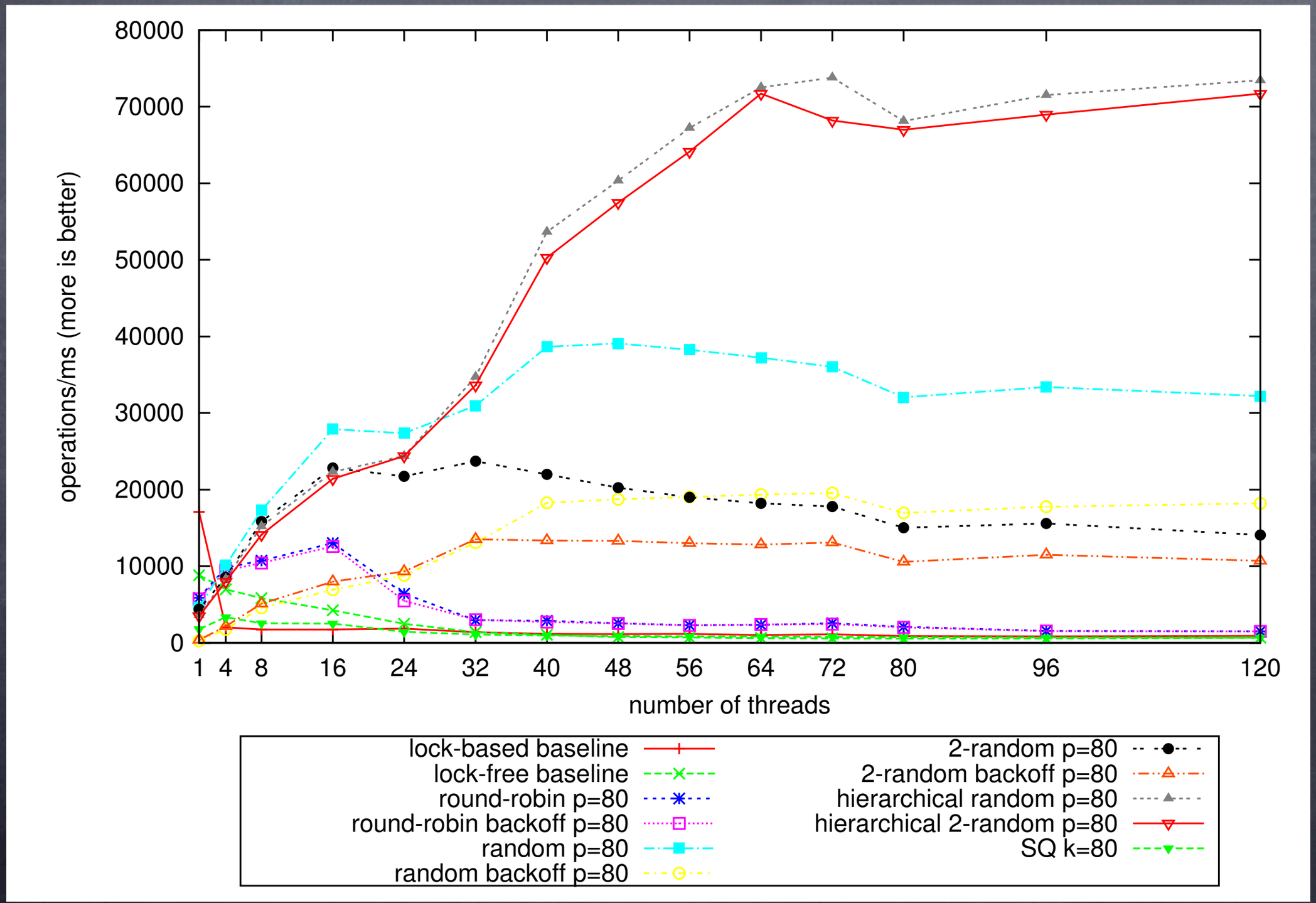
4 processors x 10 cores x 2 hardware threads = 80 hardware threads



4 partitions of p/4 partial FIFO queues each (one partition for each processor):

1. select processor-local partition with given probability w (others with 1-w)
2. select partial FIFO queue in selected partition using RA or dRA

Performance & Scalability

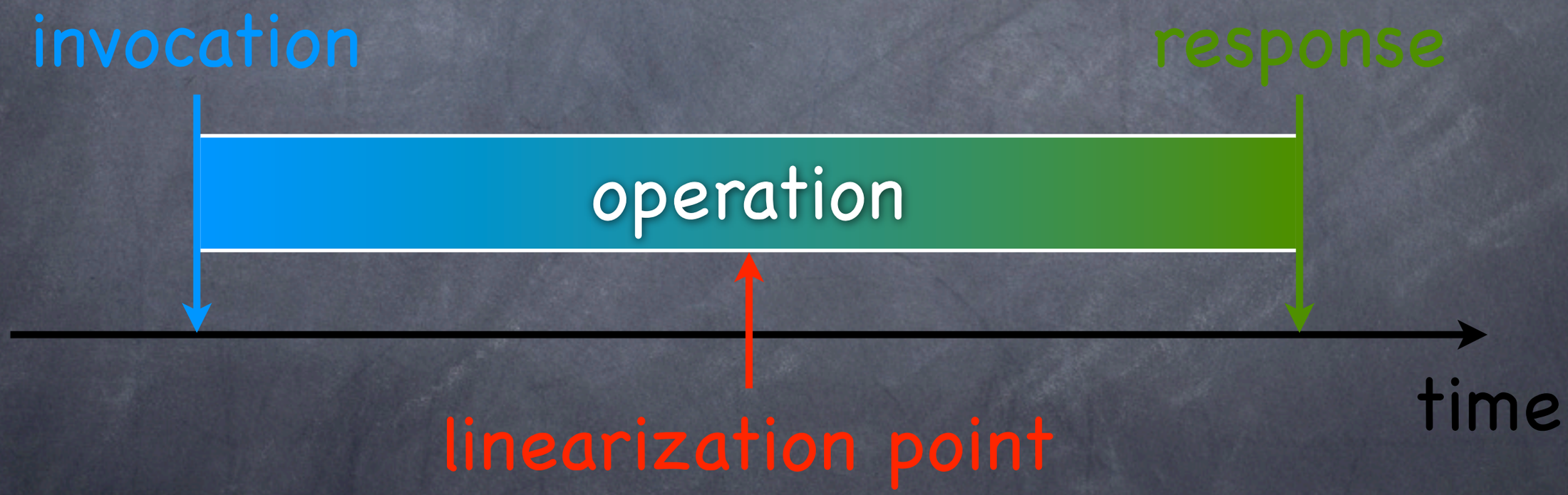


Execution History

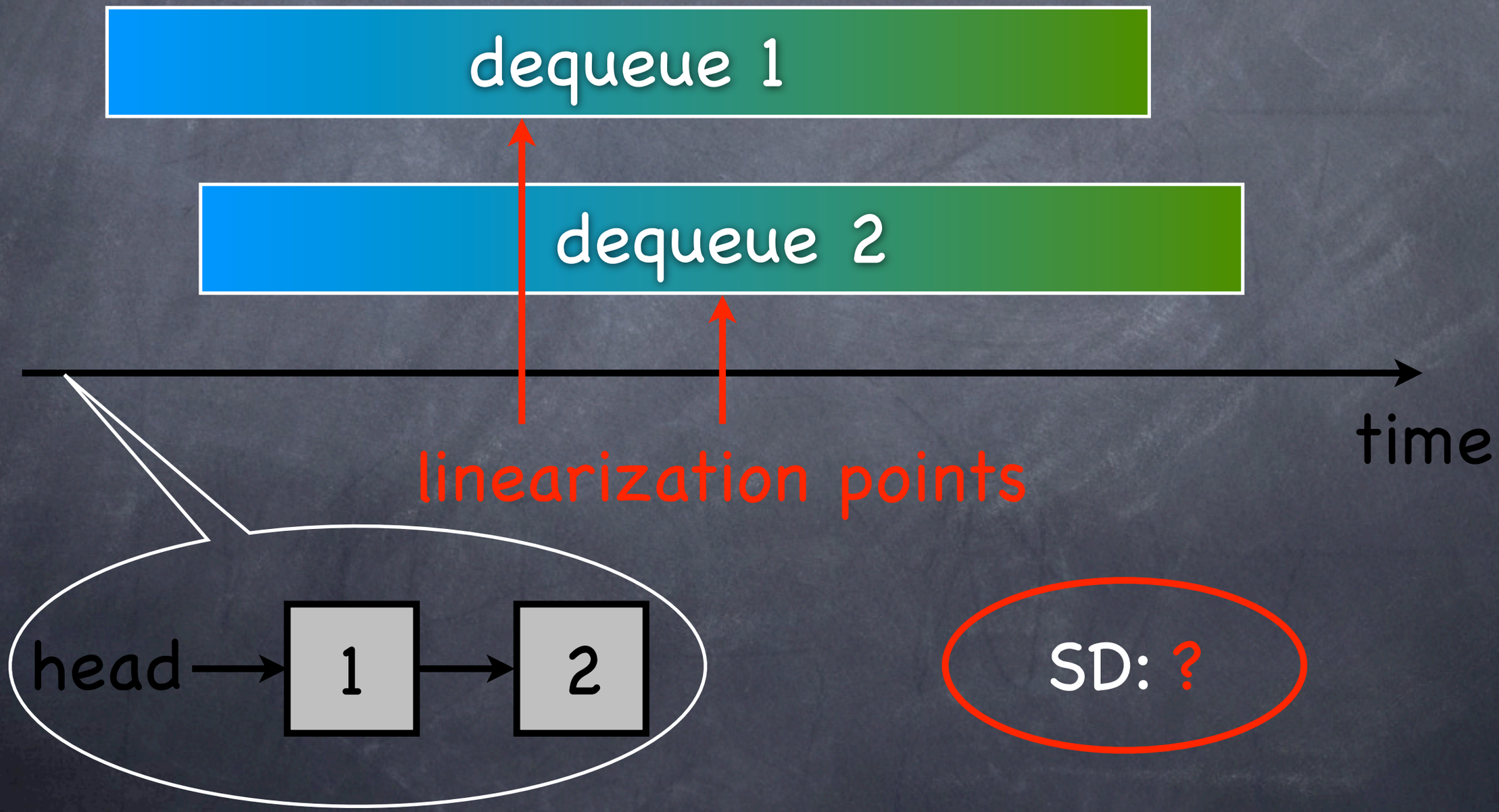
Sequence of Time-Stamped Invocation and Response Events

Execution History

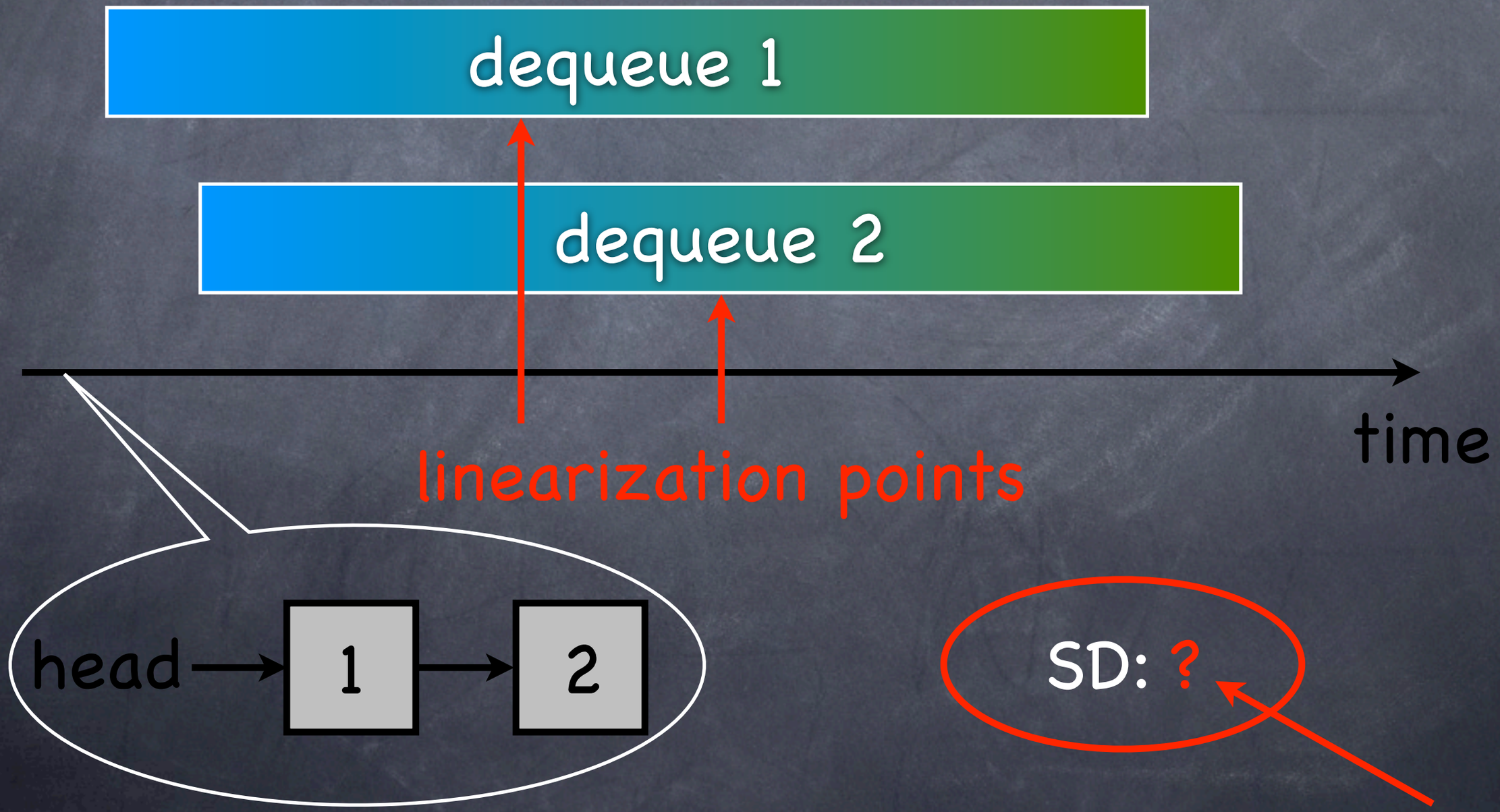
Sequence of Time-Stamped Invocation and Response Events



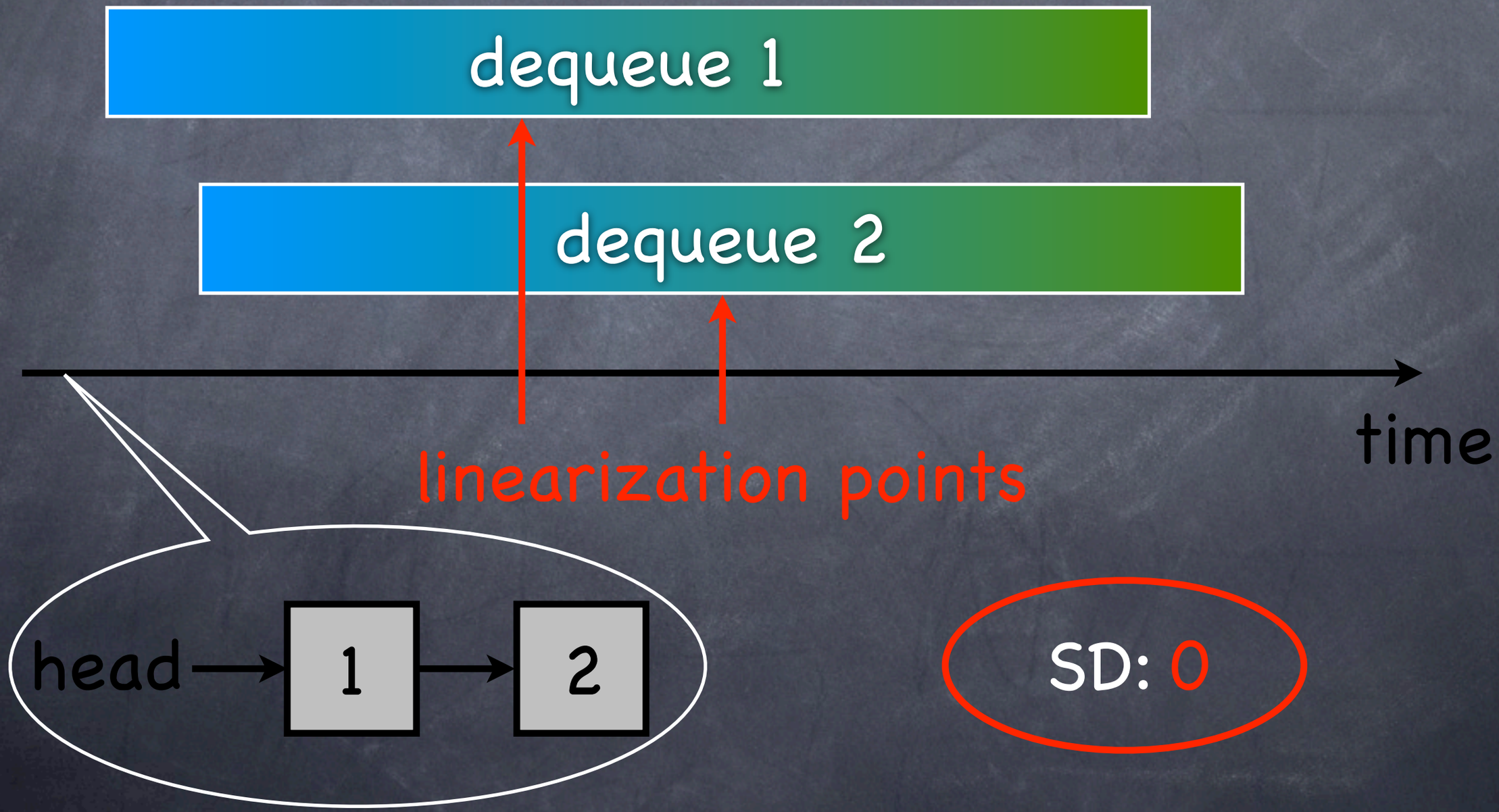
Measuring Semantical Deviation (SD)



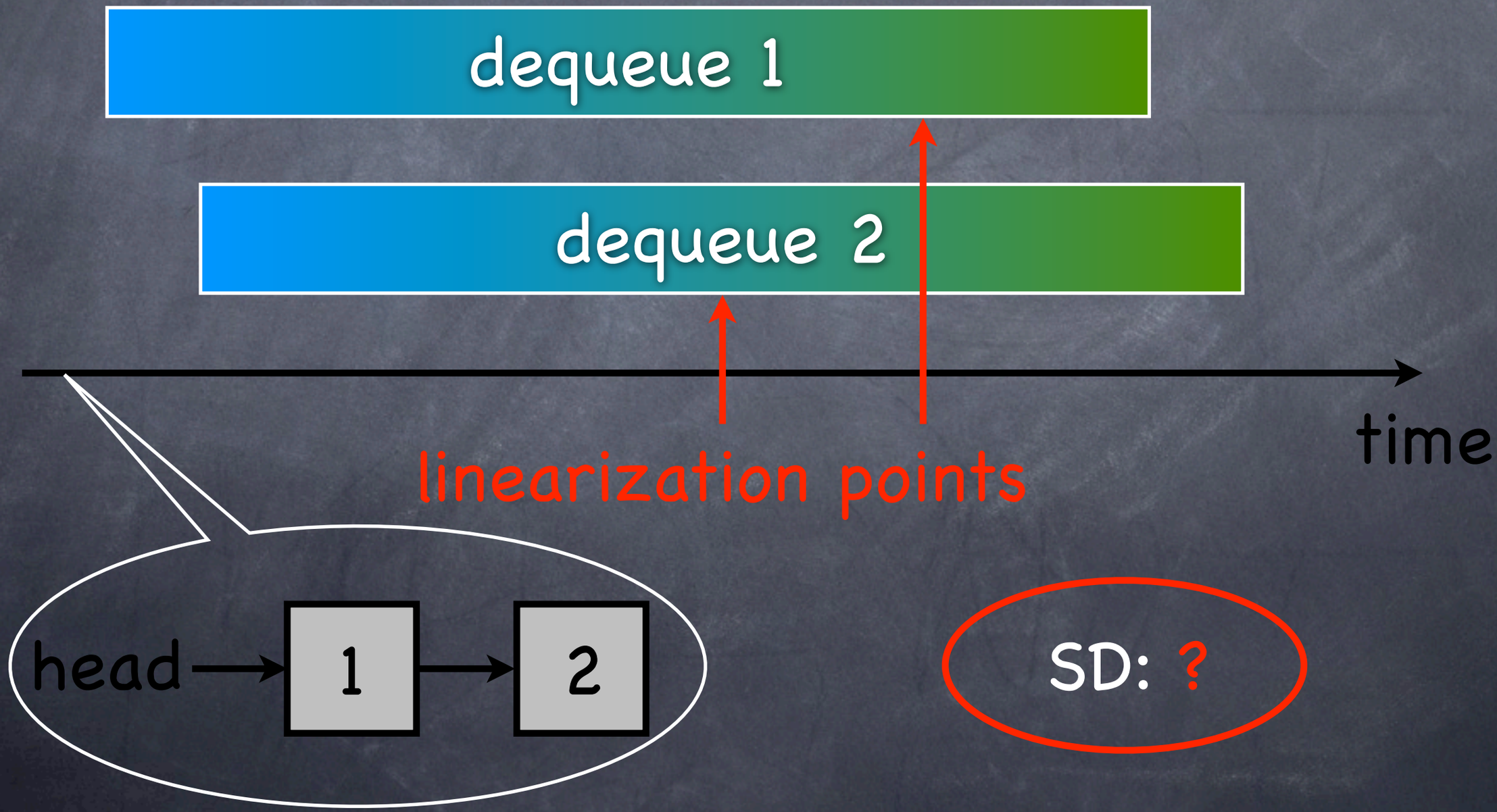
Measuring Semantical Deviation (SD)



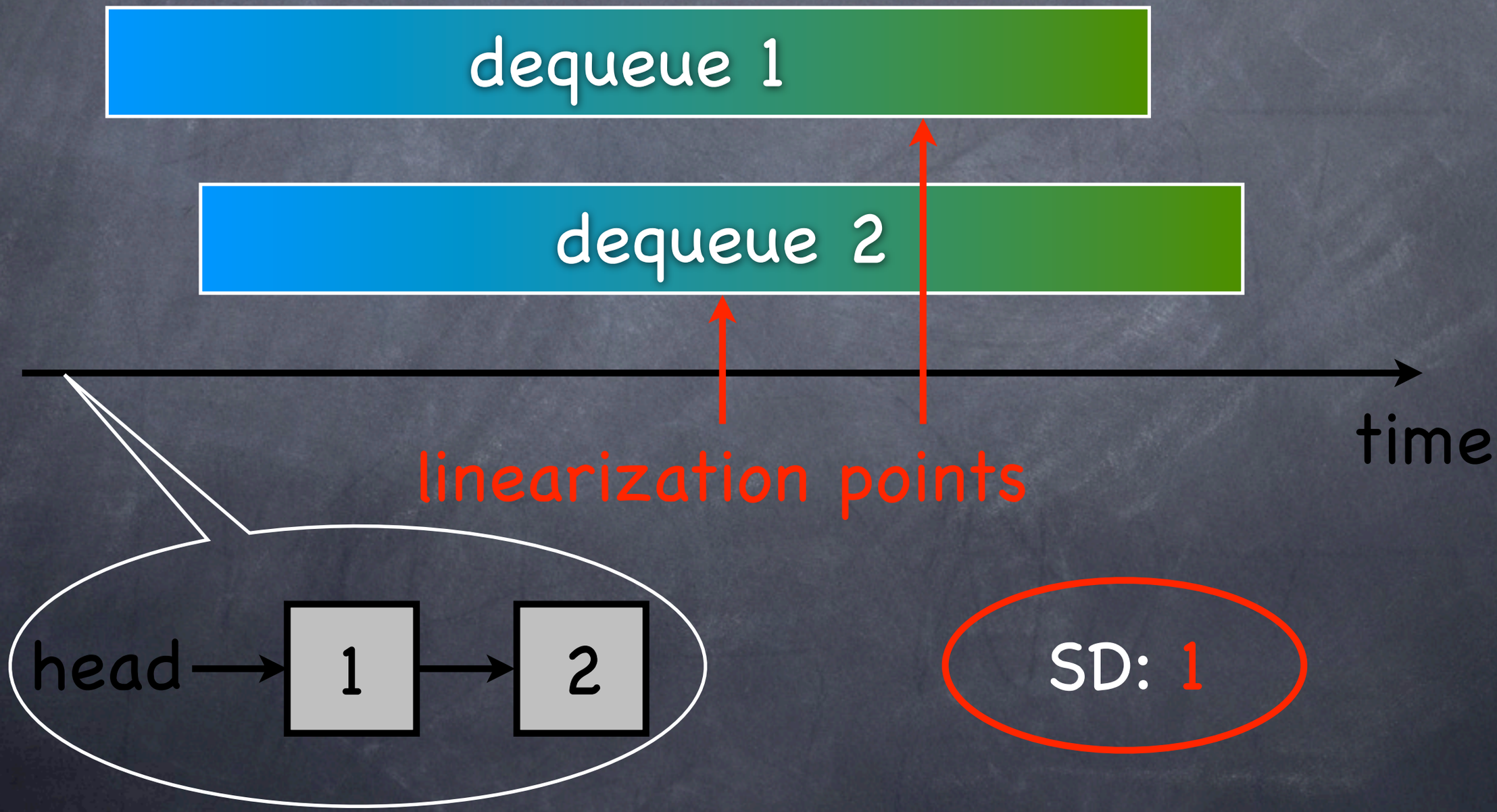
Measuring Semantical Deviation (SD)



Measuring Semantical Deviation (SD)

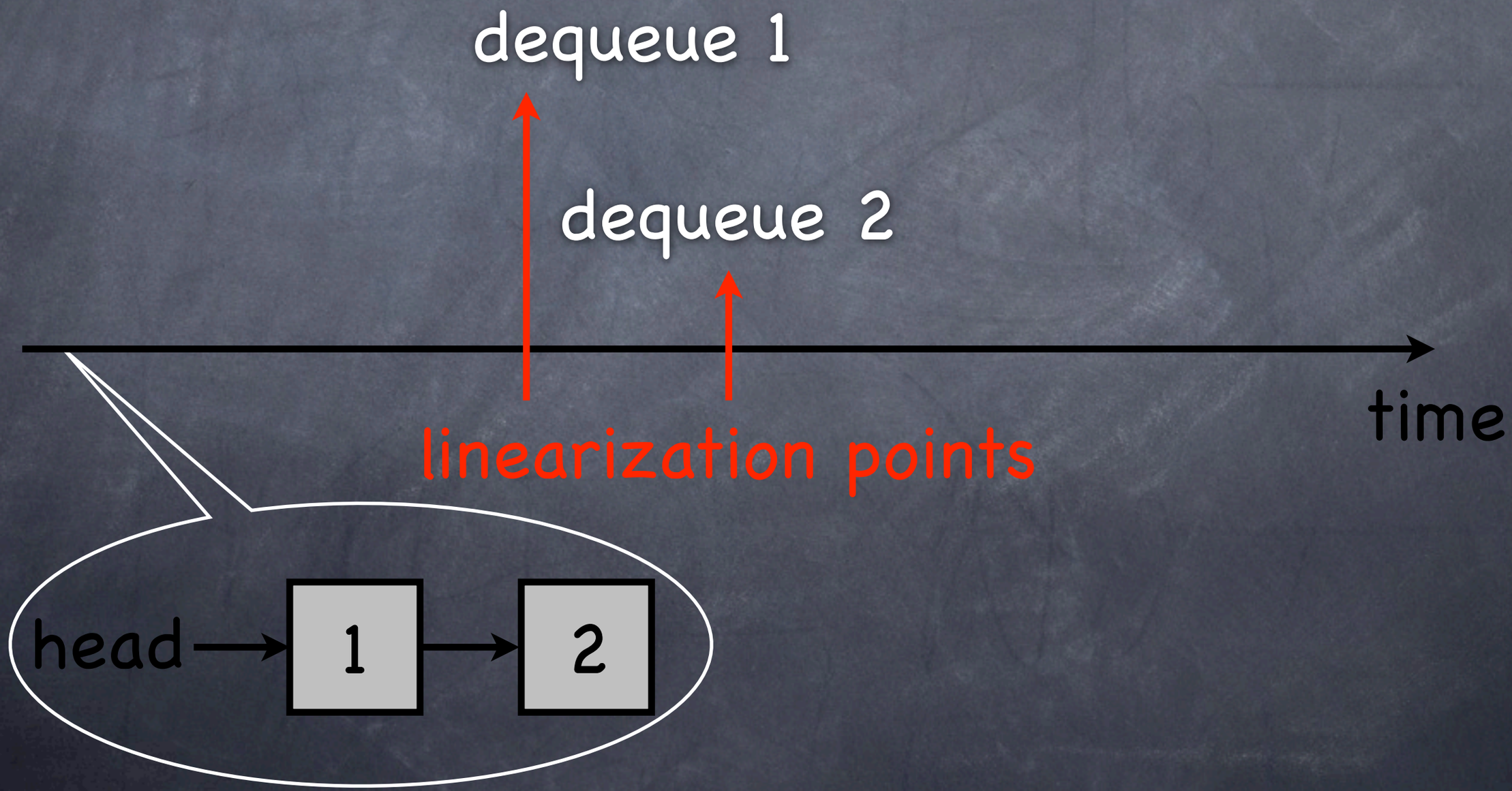


Measuring Semantical Deviation (SD)



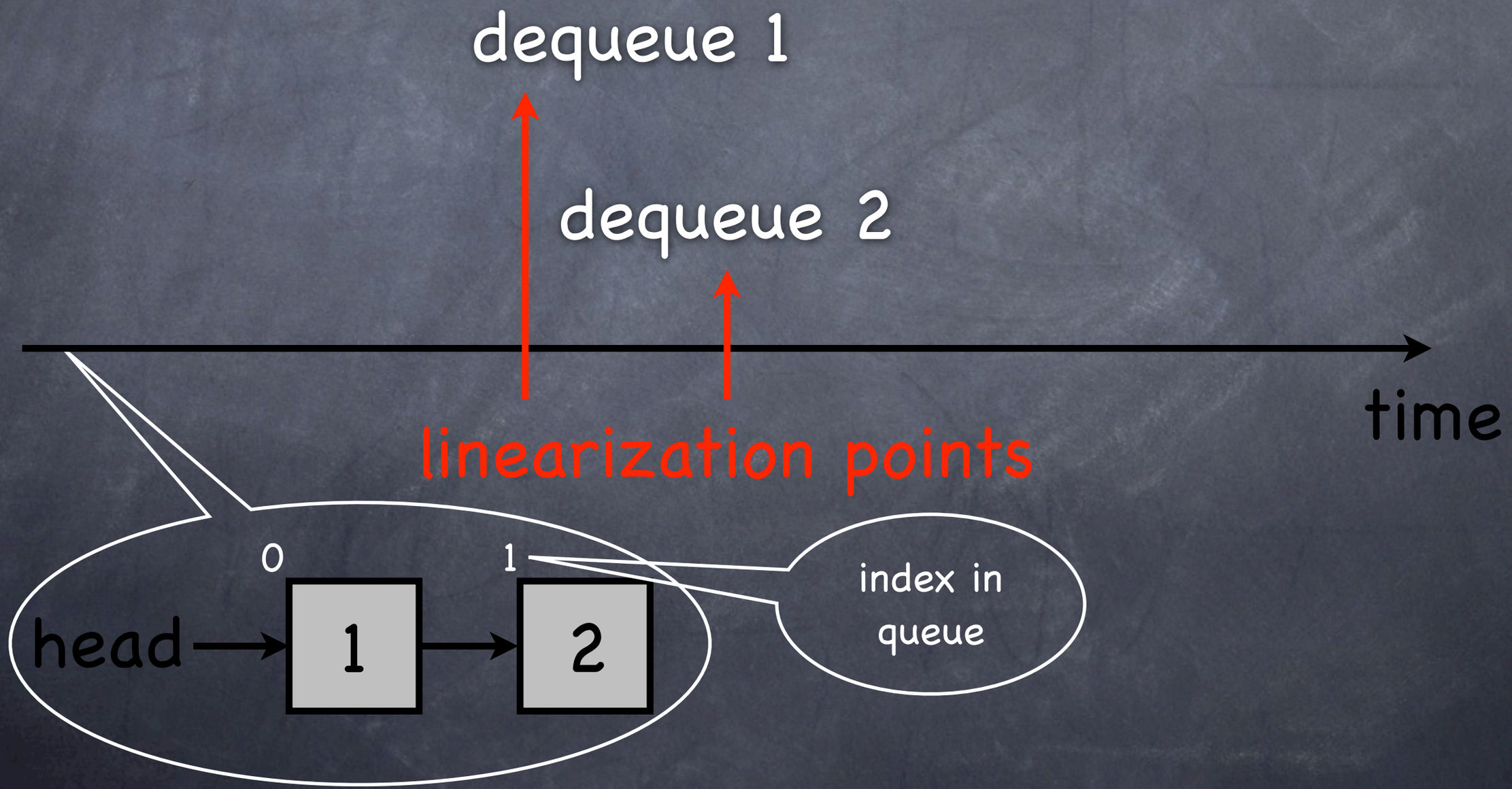
Sequential History

Sequence of Operations (Linearization Points)



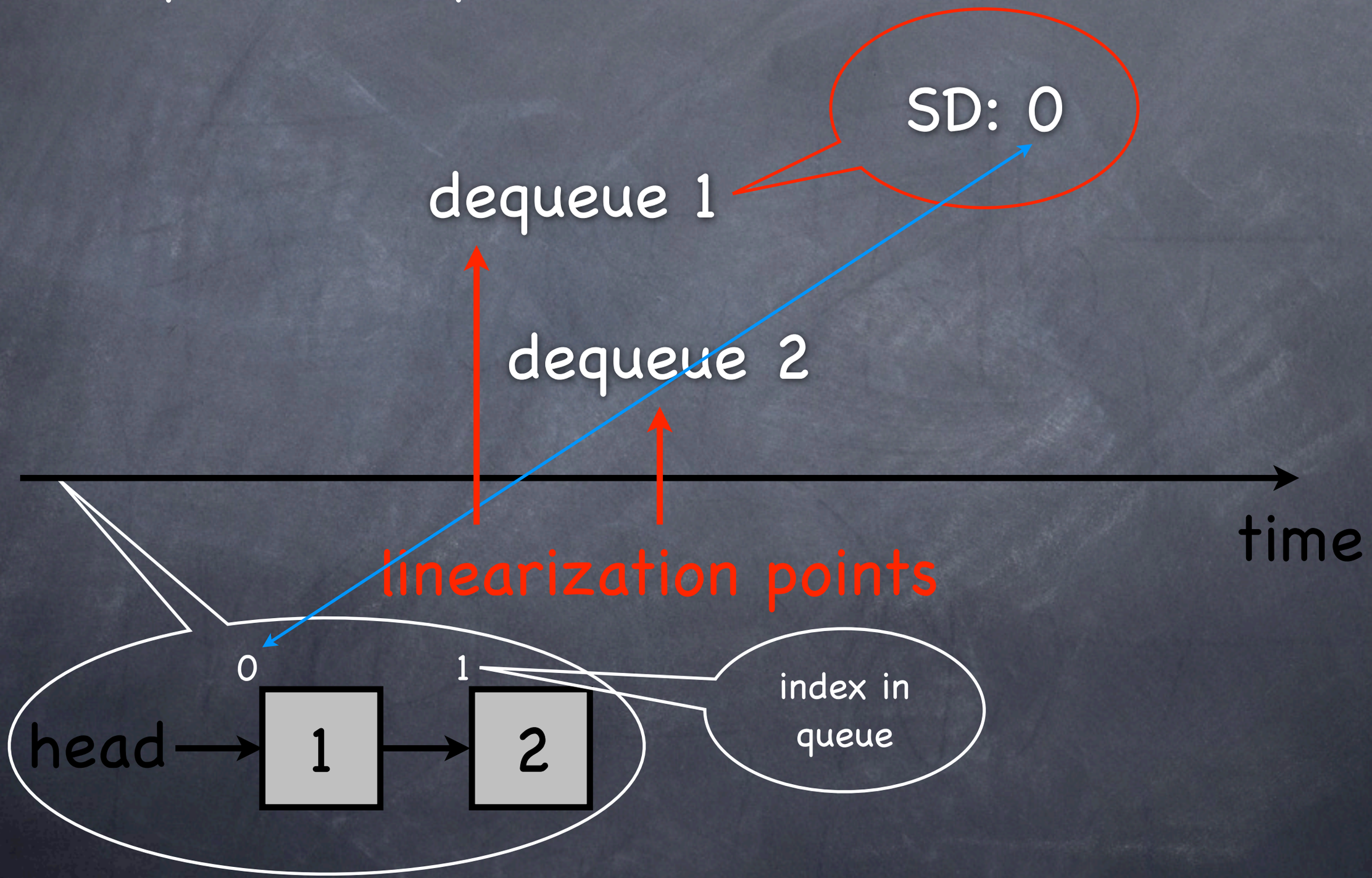
Sequential History

Sequence of Operations (Linearization Points)



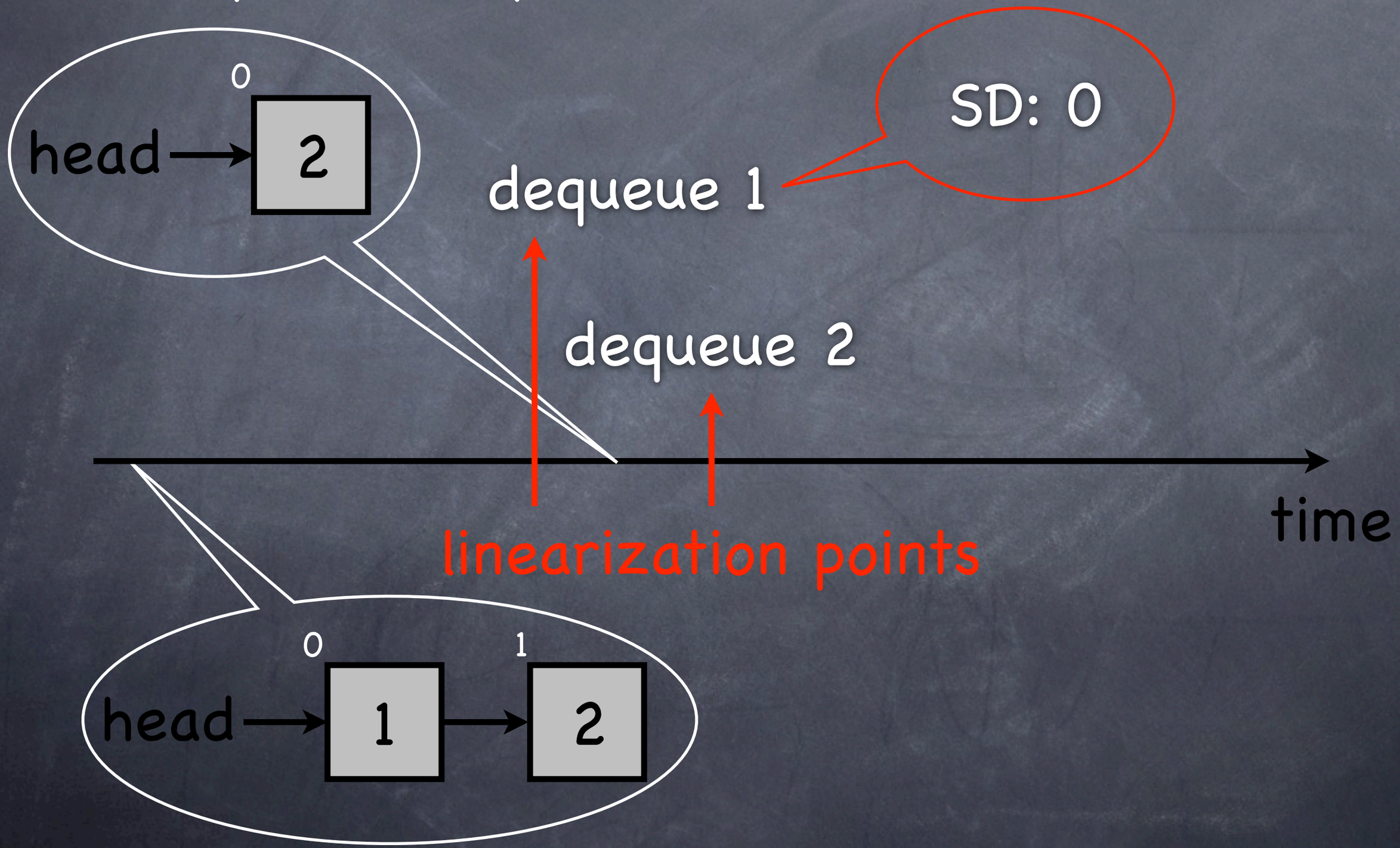
Sequential History

Sequence of Operations (Linearization Points)



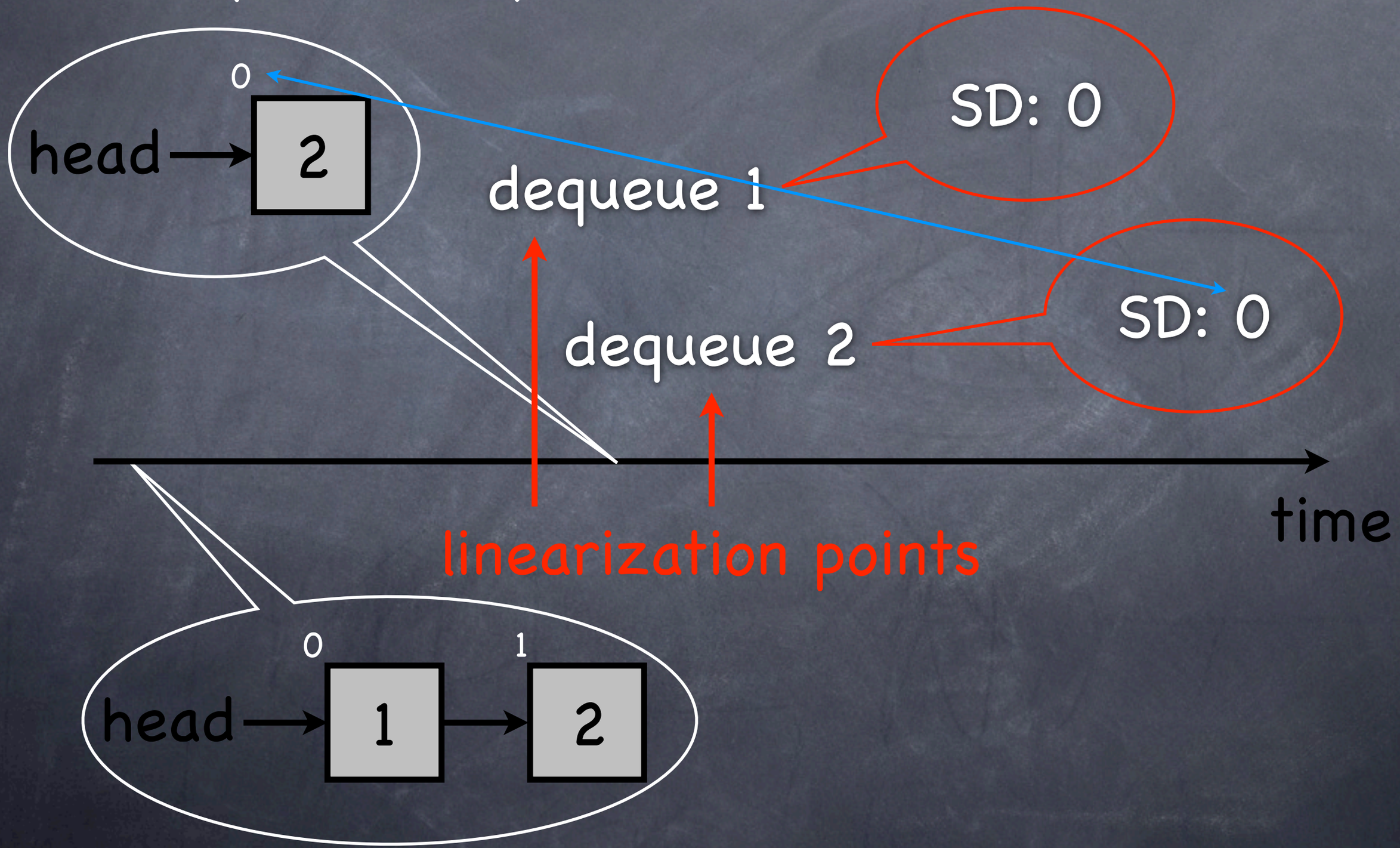
Sequential History

Sequence of Operations (Linearization Points)



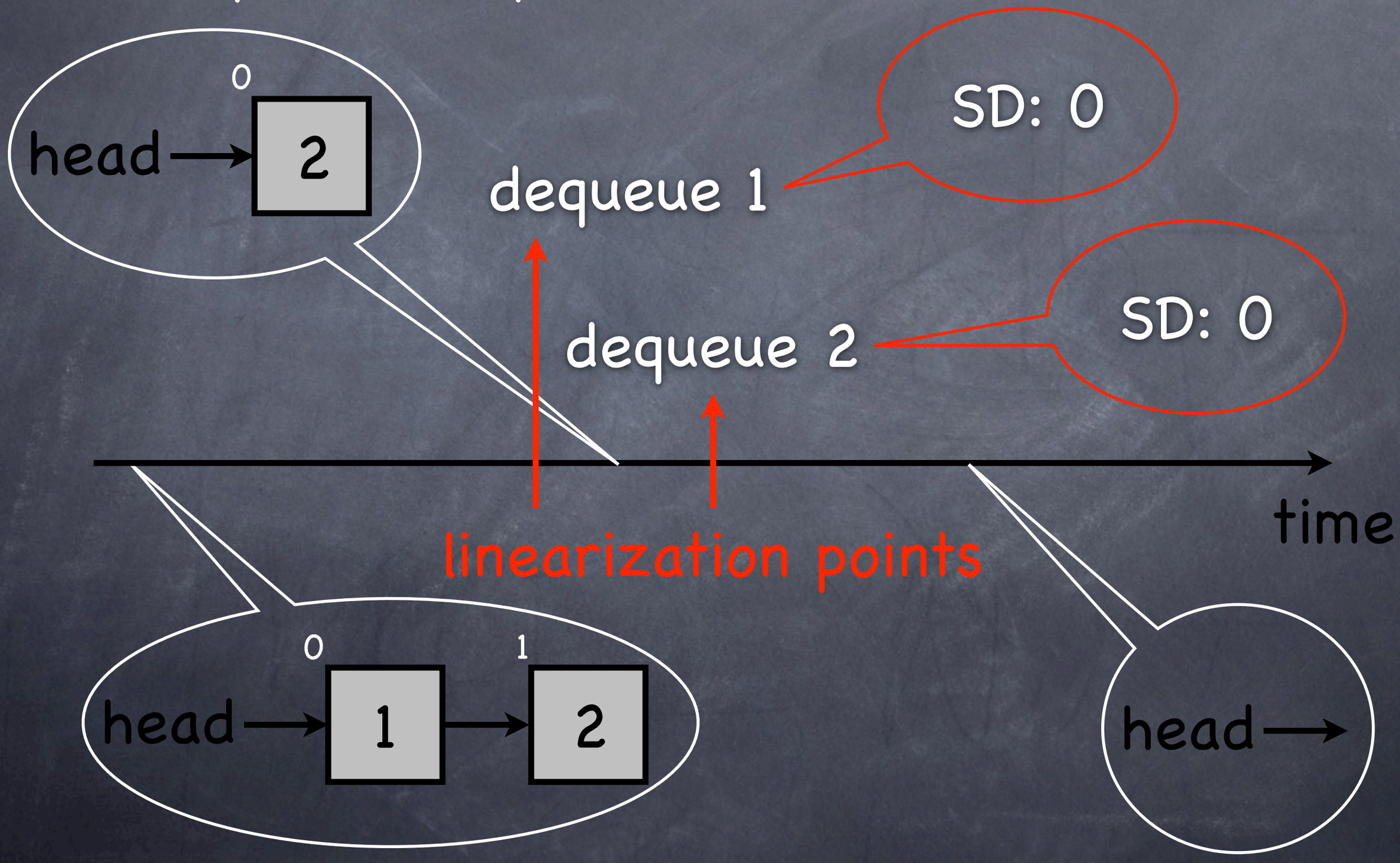
Sequential History

Sequence of Operations (Linearization Points)



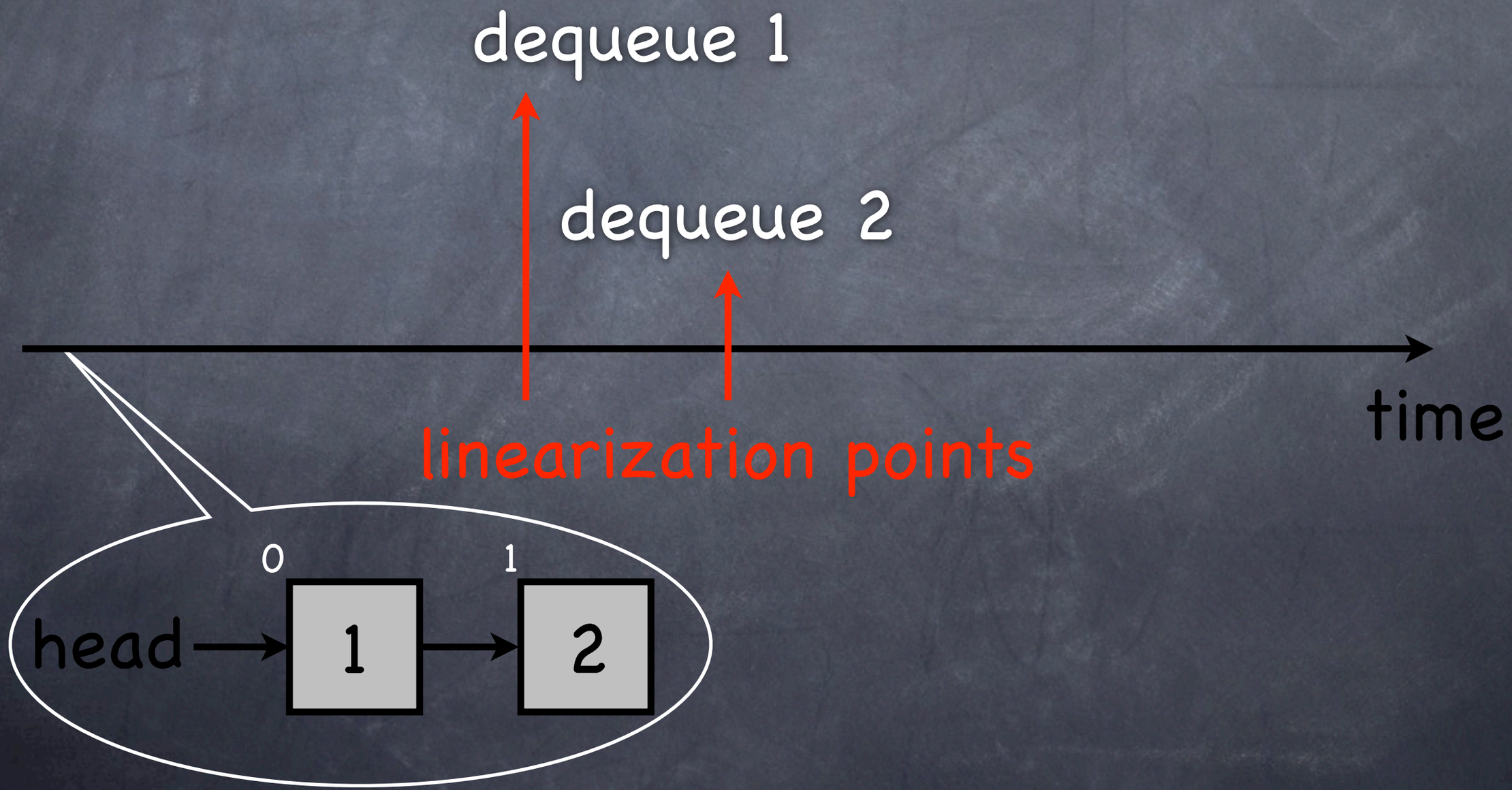
Sequential History

Sequence of Operations (Linearization Points)



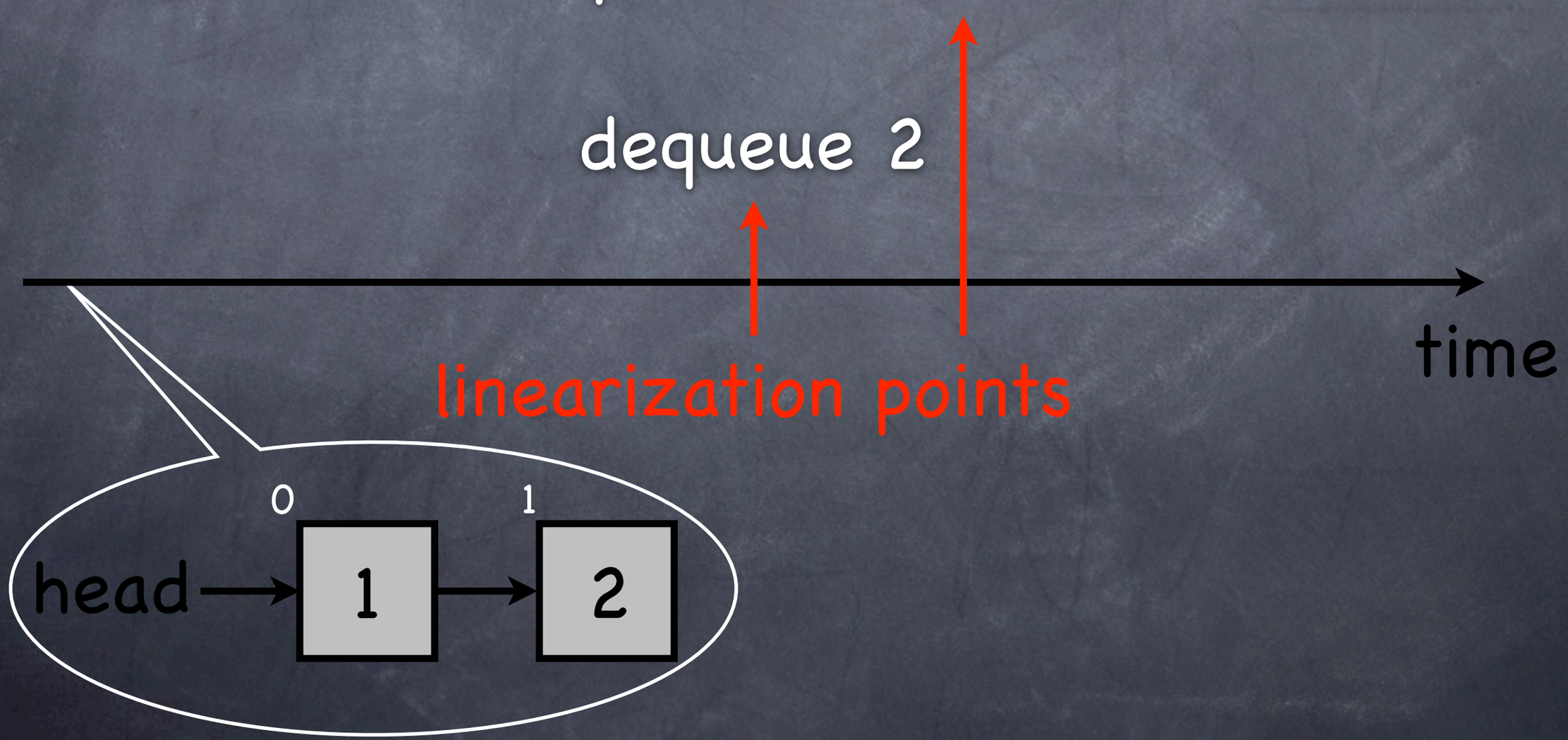
Sequential History II

Sequence of Operations (Linearization Points)



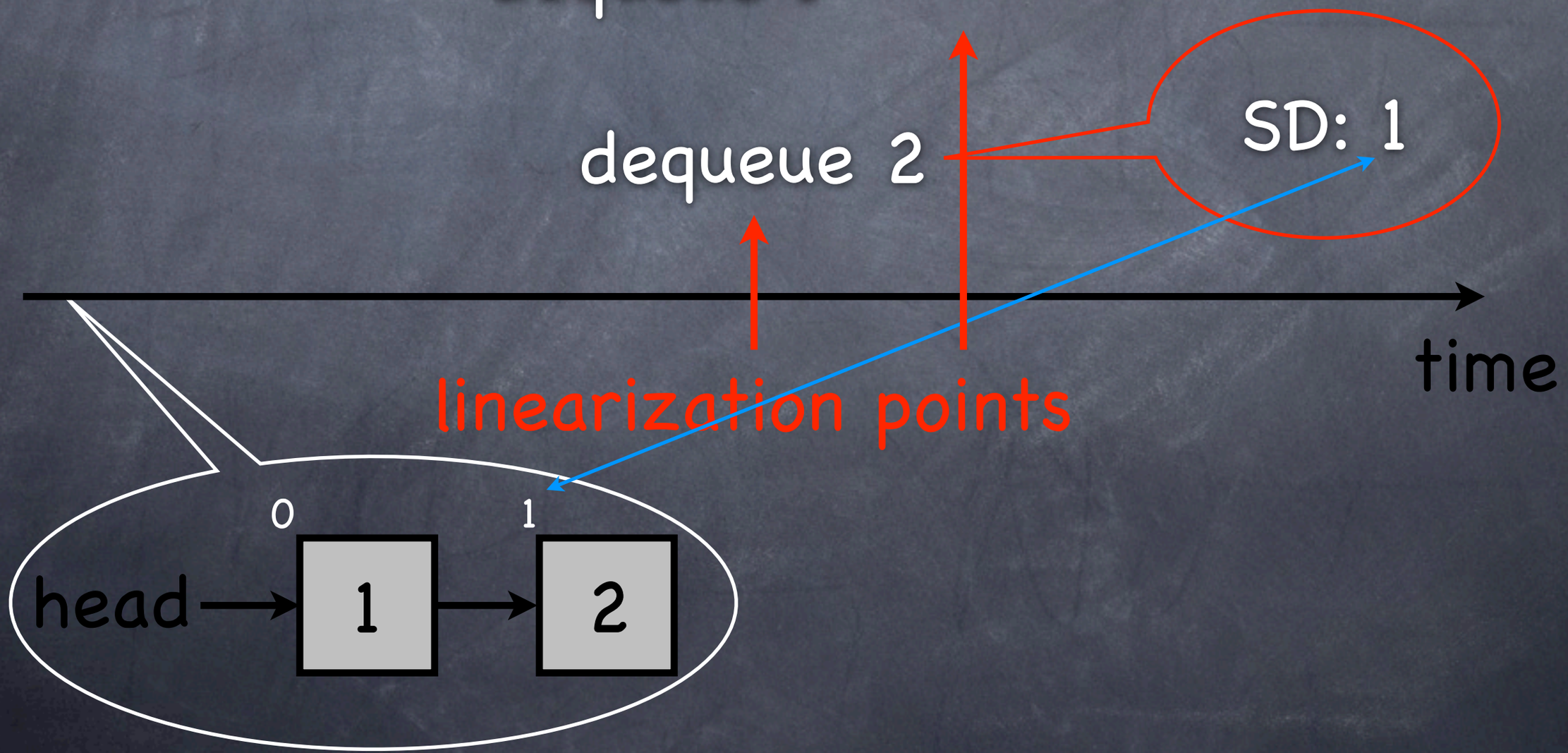
Sequential History II

Sequence of Operations (Linearization Points)



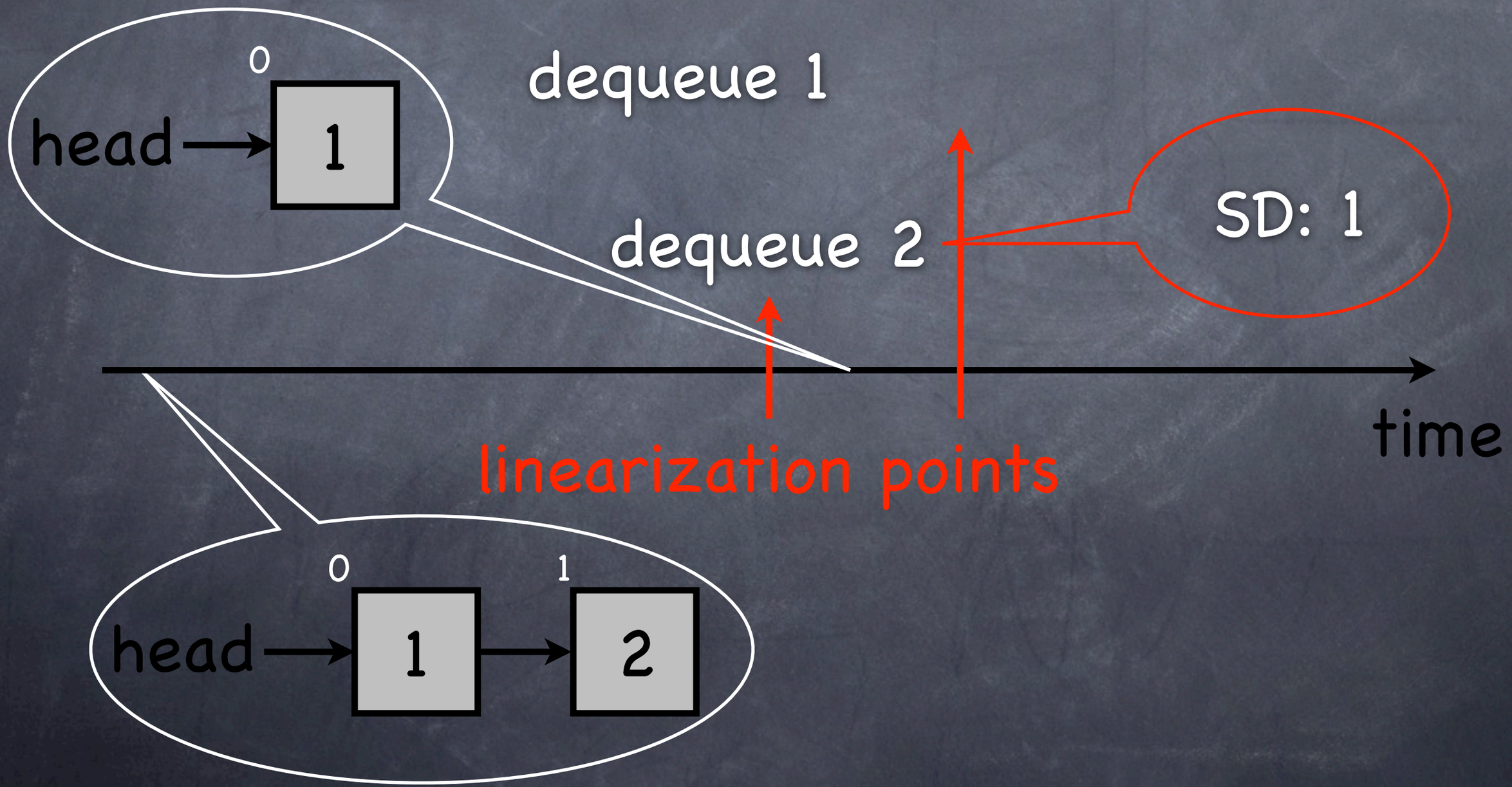
Sequential History II

Sequence of Operations (Linearization Points)



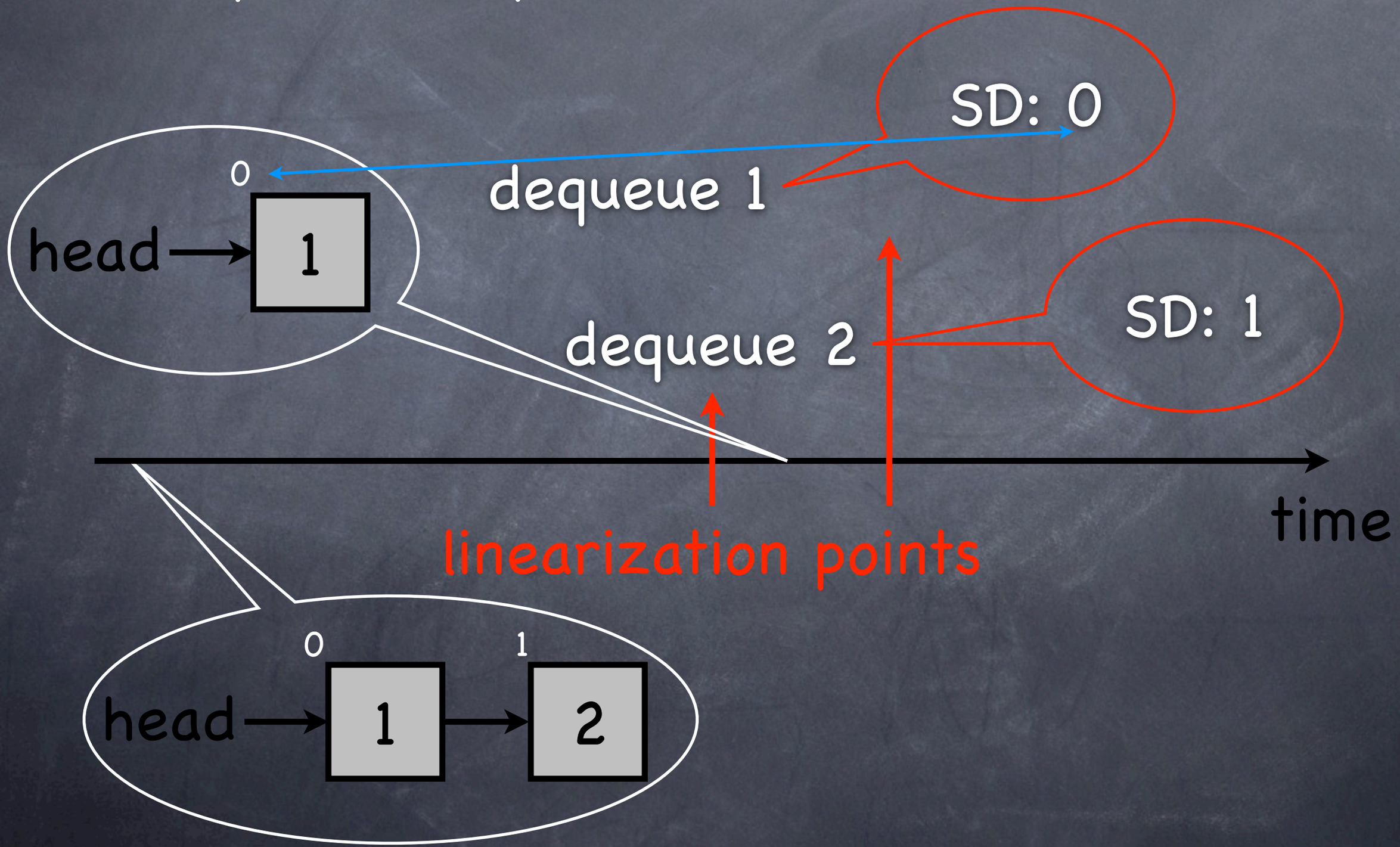
Sequential History II

Sequence of Operations (Linearization Points)



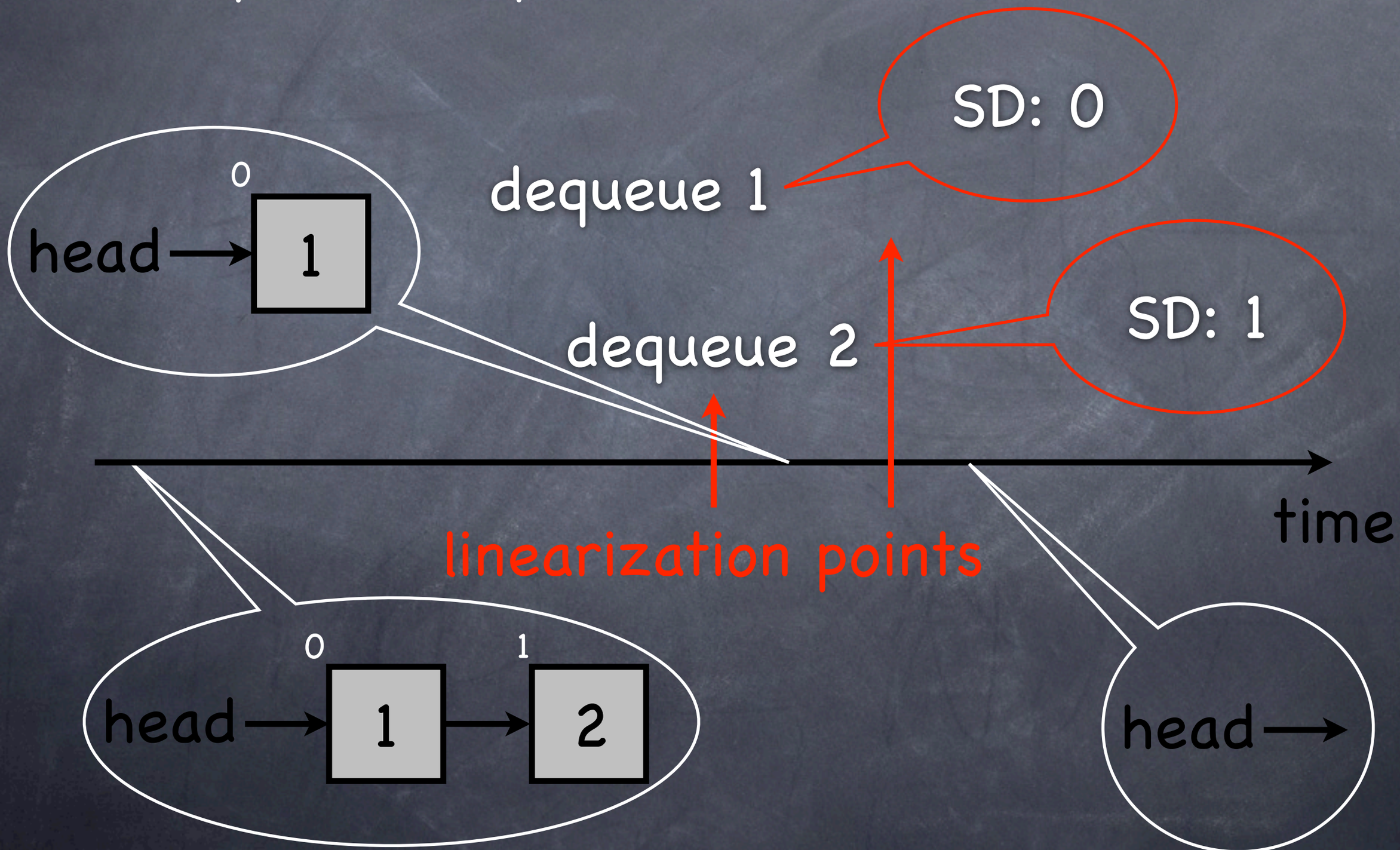
Sequential History II

Sequence of Operations (Linearization Points)



Sequential History II

Sequence of Operations (Linearization Points)



The semantical deviation (SD) of a sequential history is the maximum of the semantical deviations of all operations of that history

Actual Semantical Deviation (ASD)

- ASD is the semantical deviation of the (generally unknown) sequential history that *actually* took place

Actual Semantical Deviation (ASD)

- ASD is the semantical deviation of the (generally unknown) sequential history that actually took place
- ASD denotes the semantical deviation of a k-FIFO queue implementation when applied to a given workload

Actual Semantical Deviation (ASD)

- ASD is the semantical deviation of the (generally unknown) sequential history that actually took place
- ASD denotes the semantical deviation of a k-FIFO queue implementation when applied to a given workload
- ASD can in general not be determined exactly, only approximated

ASD Analysis

First Attempt

1. Run a k-FIFO queue implementation on a given workload and obtain execution history

ASD Analysis

First Attempt

1. Run a k-FIFO queue implementation on a given workload and obtain execution history
2. Determine the sequential histories with the lowest (LSD) and the highest semantical deviation (HSD) among all sequential histories of the execution history

ASD Analysis

First Attempt

1. Run a k-FIFO queue implementation on a given workload and obtain execution history
2. Determine the sequential histories with the lowest (LSD) and the highest semantical deviation (HSD) among all sequential histories of the execution history

Then: LSD ≤ ASD ≤ HSD

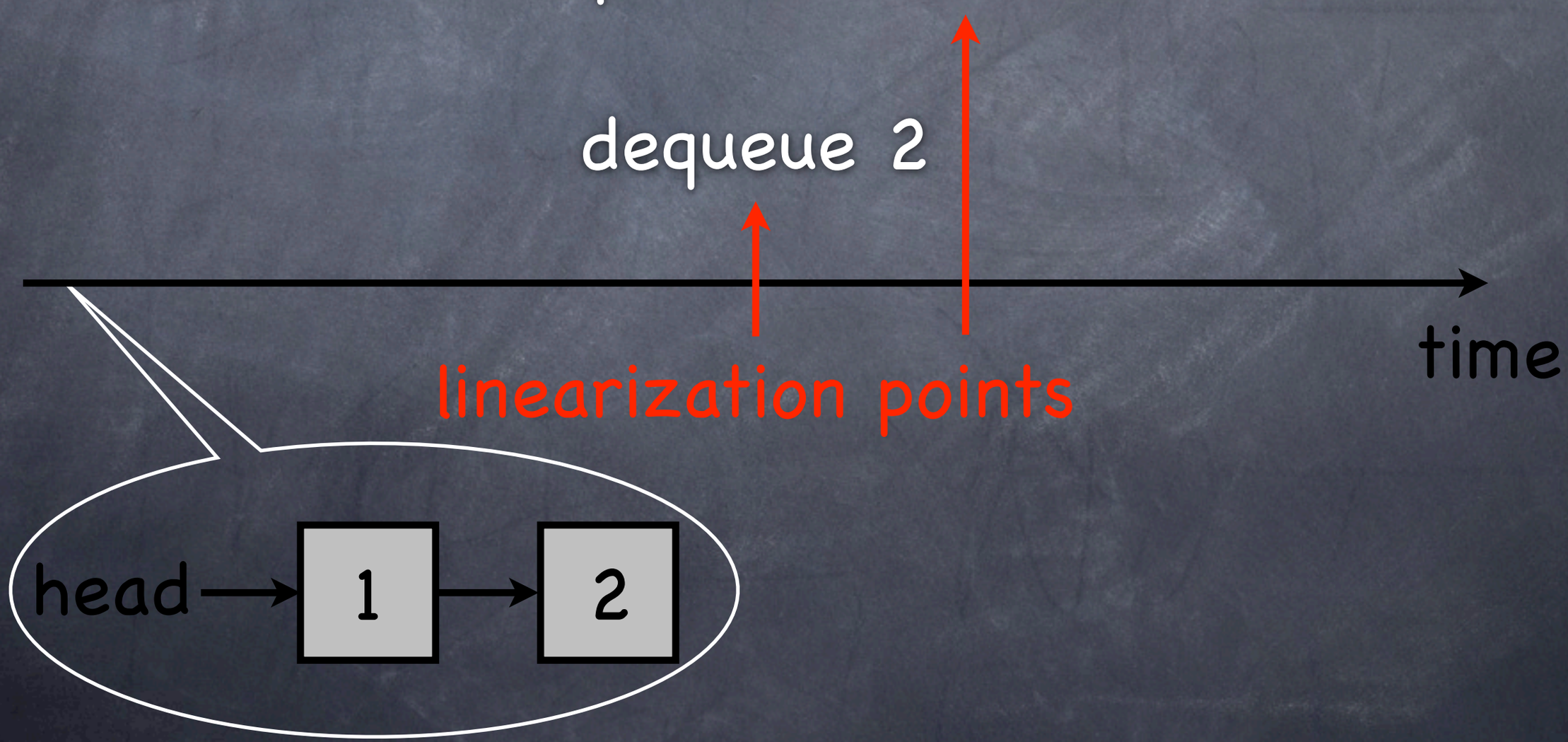
ASD Analysis

First Attempt

1. Run a k-FIFO queue implementation on a given workload and obtain execution history
2. Determine the sequential histories with the lowest (LSD) and the highest semantical deviation (HSD) among all sequential histories of the execution history

But: HSD ≤ WCSD may not hold

Invalid Sequential History (if k=0)



ASD Analysis

For small WCSD

1. Run a k-FIFO queue implementation on a given workload and obtain execution history
2. Determine the sequential histories with the lowest (LSD) and the highest semantical deviation (HSD) among all valid sequential histories of the execution history

ASD Analysis

For small WCSD

1. Run a k-FIFO queue implementation on a given workload and obtain execution history
2. Determine the sequential histories with the lowest (LSD) and the highest semantical deviation (HSD) among all valid sequential histories of the execution history

But what if WCSD is large or ∞ ?

ASD Analysis

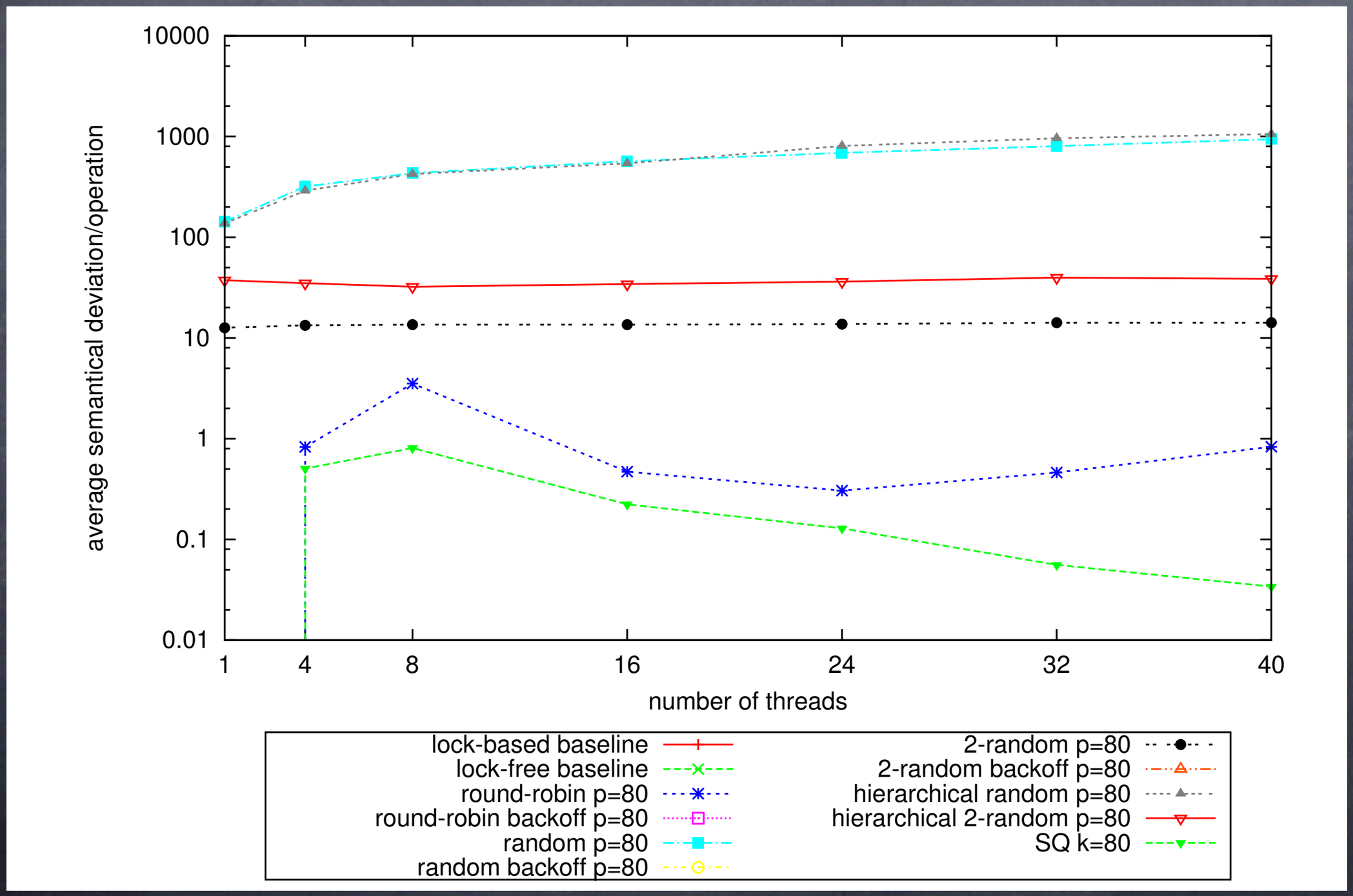
Current Version

1. Run a k-FIFO queue implementation on a given workload and obtain execution history

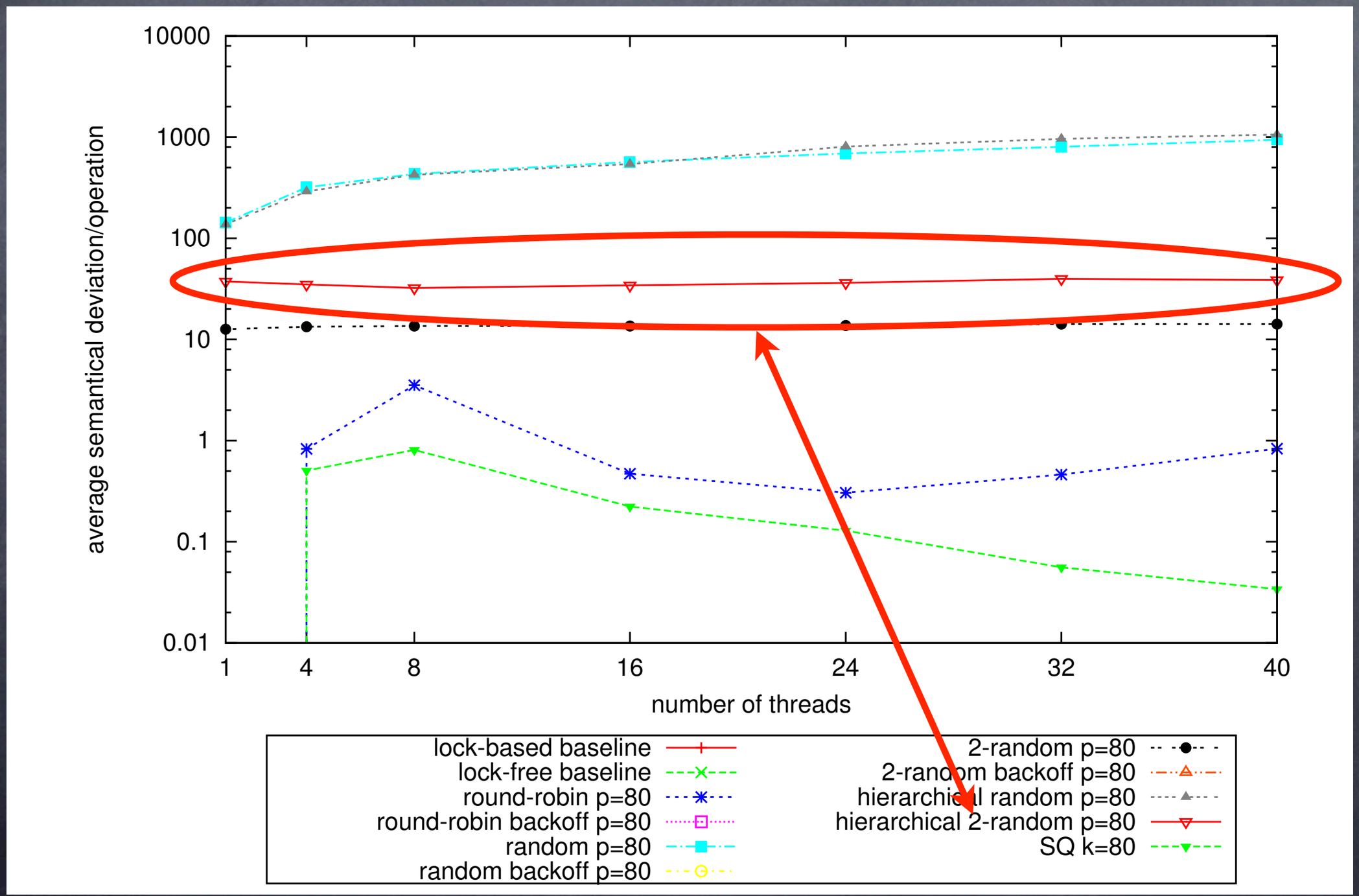
2. Determine the sequential history with the lowest semantical deviation (LSD) among all valid sequential histories of the execution history

3. Depict the average of the semantical deviation of the operations in that sequential history

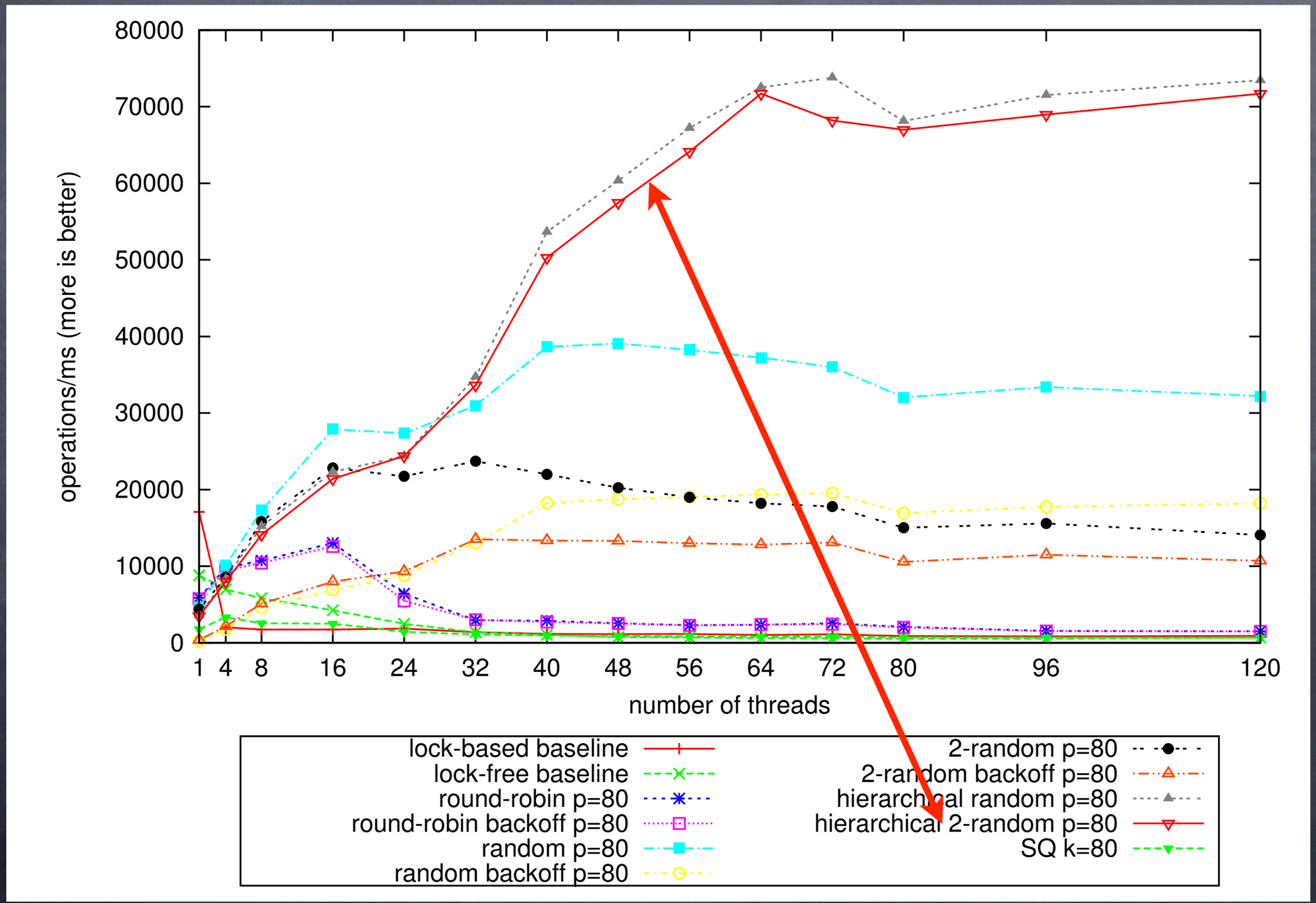
Average Semantical Deviation of LSD History



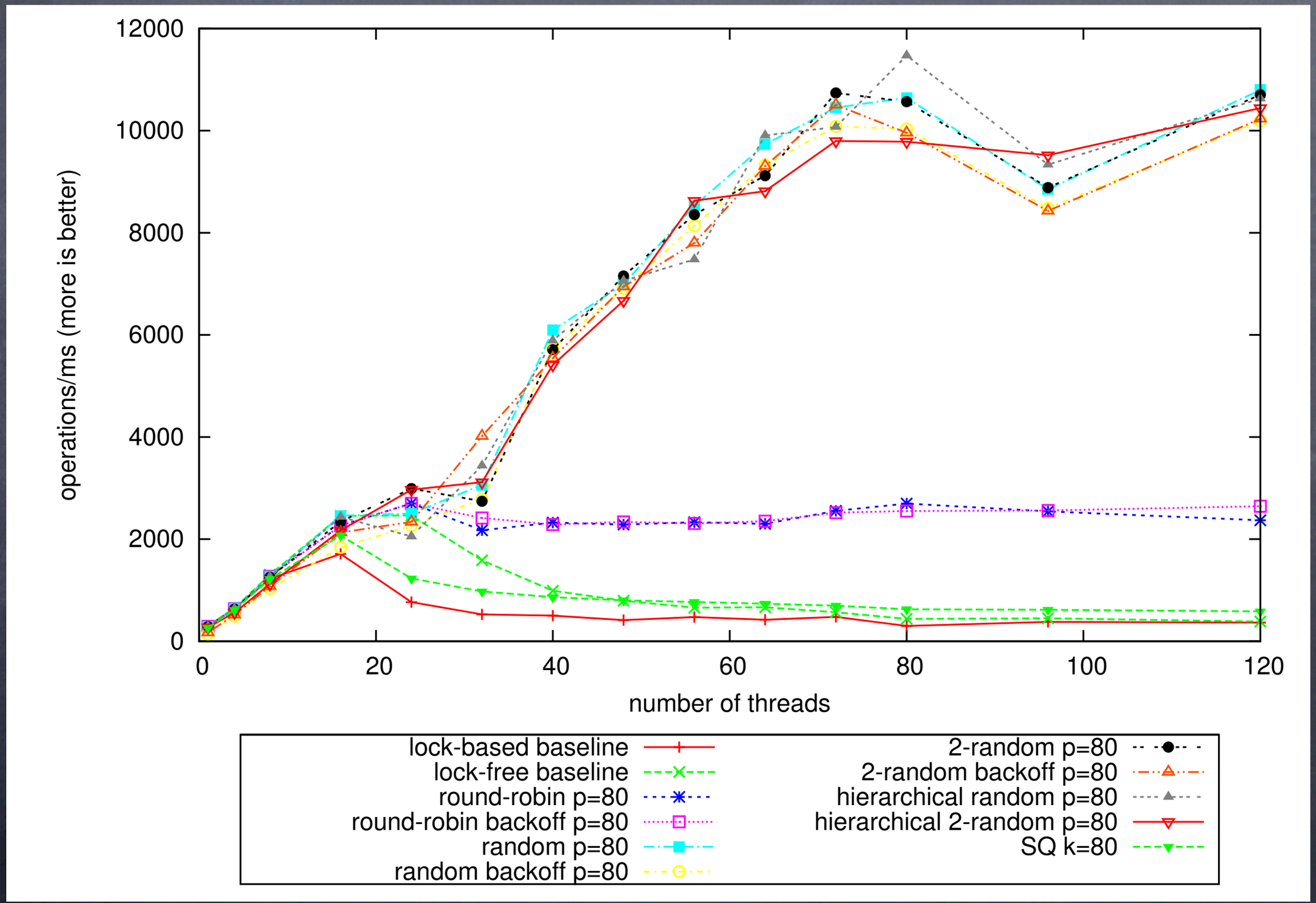
Best Trade-off



Hierarchical 2-Random



Low Contention



Performance-aware Programming

Future programming paradigms will need to incorporate performance as first-class concept!

But should we expose the machine architecture, in particular the memory hierarchy to the programmer?

Future Work

- Using k-FIFO queues in applications:
  - e.g. to construct concurrent and scalable real-time schedulers for multicore systems

Future Work

- Using k-FIFO queues in applications:
  - e.g. to construct concurrent and scalable real-time schedulers for multicore systems
- Formally proving WCSD for given algorithms:
  - we have done this manually and informally

Future Work

- Using k-FIFO queues in applications:
  - e.g. to construct concurrent and scalable real-time schedulers for multicore systems
- Formally proving WCSD for given algorithms:
  - we have done this manually and informally
- Introducing WCSD to other data structures:
  - stacks, priority queues, hashtables, STM, ...

Thank you