Virtualizing Time, Space, and Power for Cyber-Physical Cloud Computing

Silviu Craciunas, Andreas Haas,
Christoph Kirsch, Florian Landolt,
Hannes Payer, Harald Röck,
Andreas Rottmann, Ana Sokolova,
Rainer Trummer

Universität Salzburg

Joshua Love
Raja Sengupta

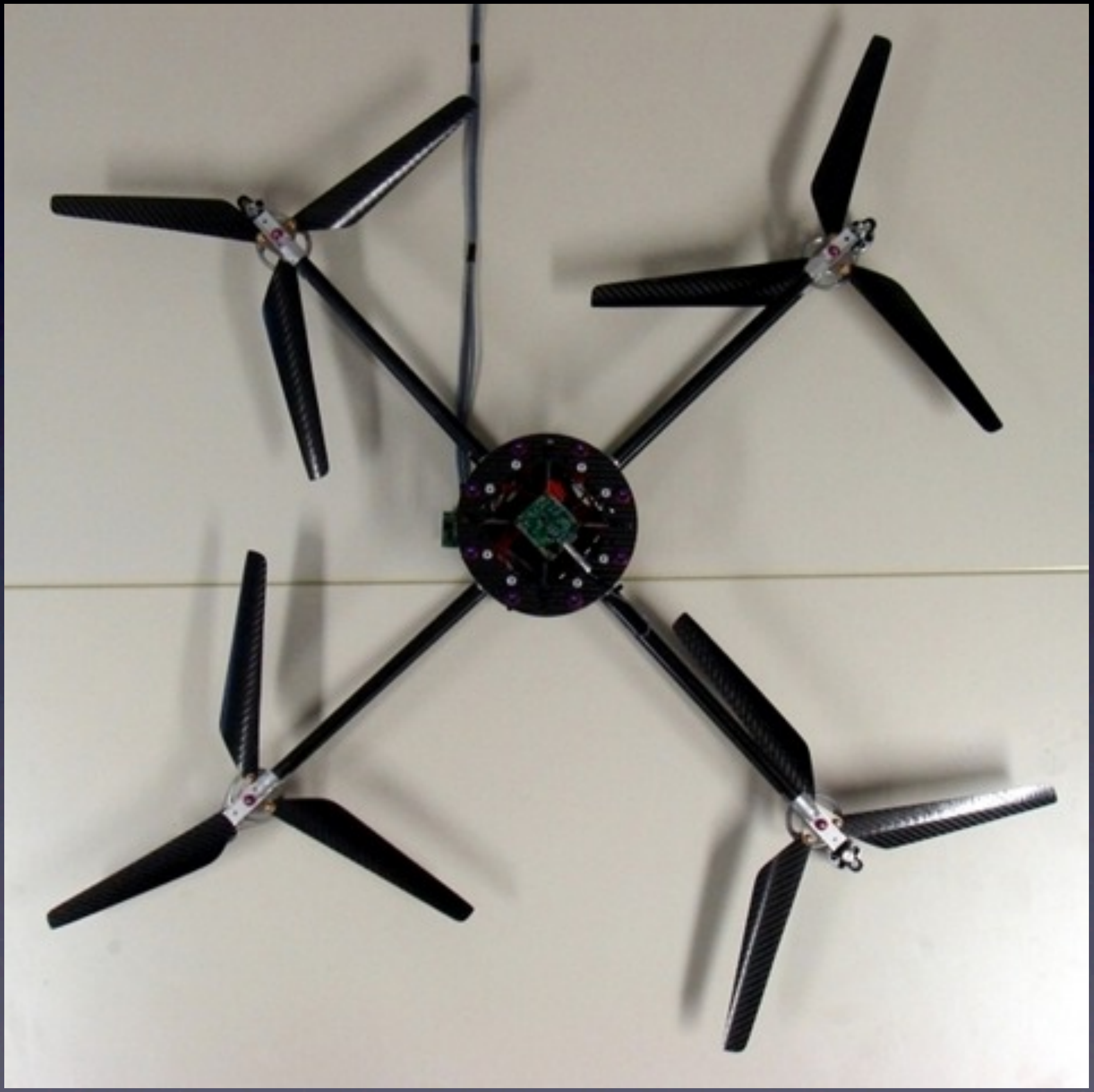
UC Berkeley

CPS Summer School, Georgia Tech, Atlanta, June 2011

The JAviator

javiator.cs.uni-salzburg.at

Quad-Rotor Helicopter



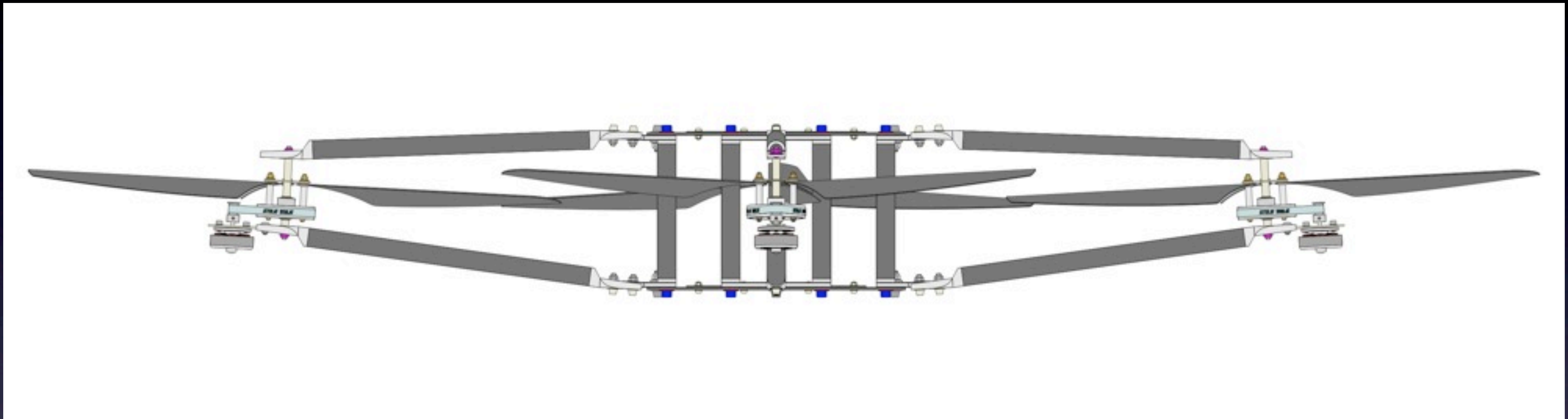
[AIAA GNC 2008]

- all carbon, titanium, aluminum design
- custom motors

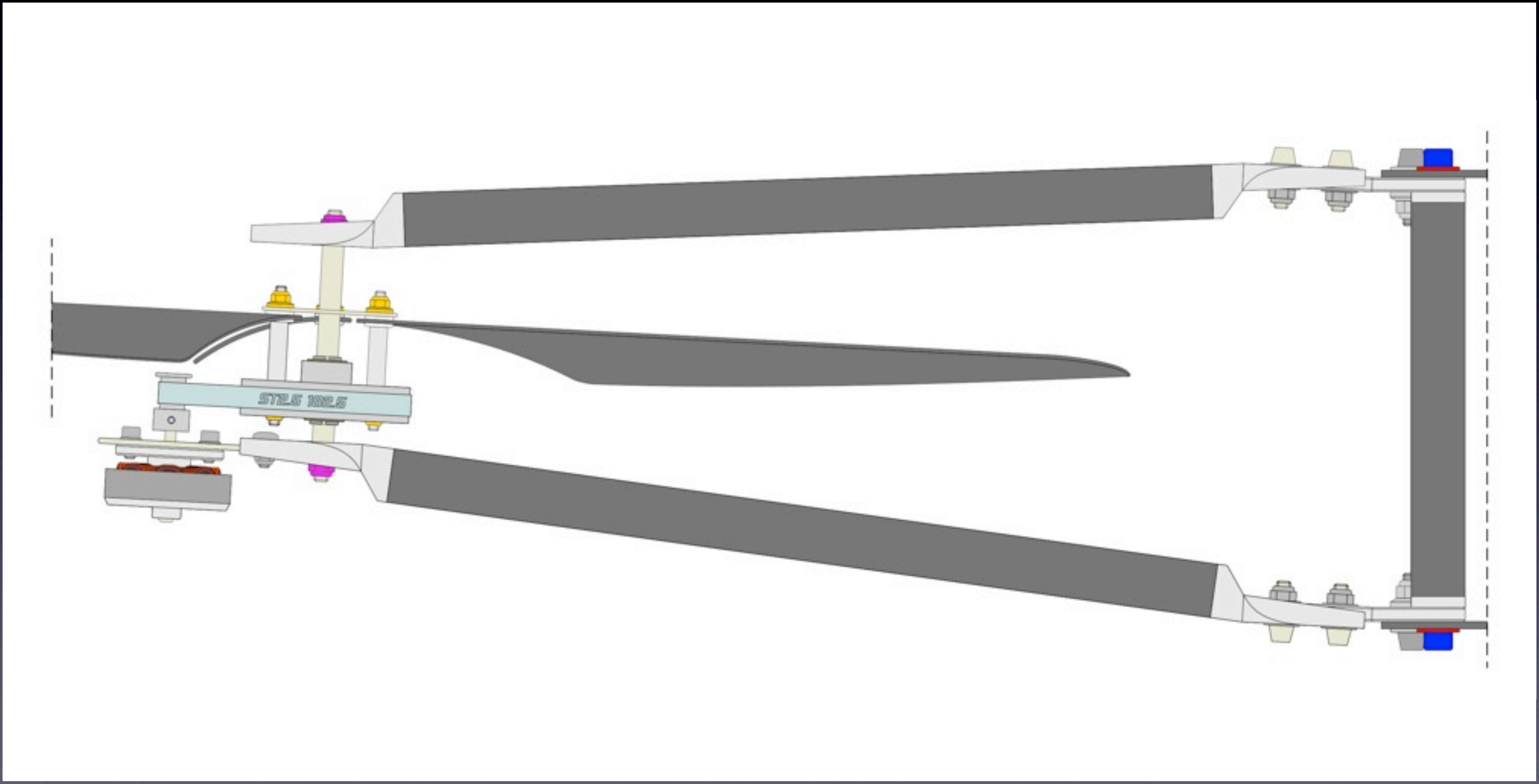
- 1.3m diameter
- ~2.2kg weight
- +2kg payload

- ~40min (empty)
- ~10min (full)

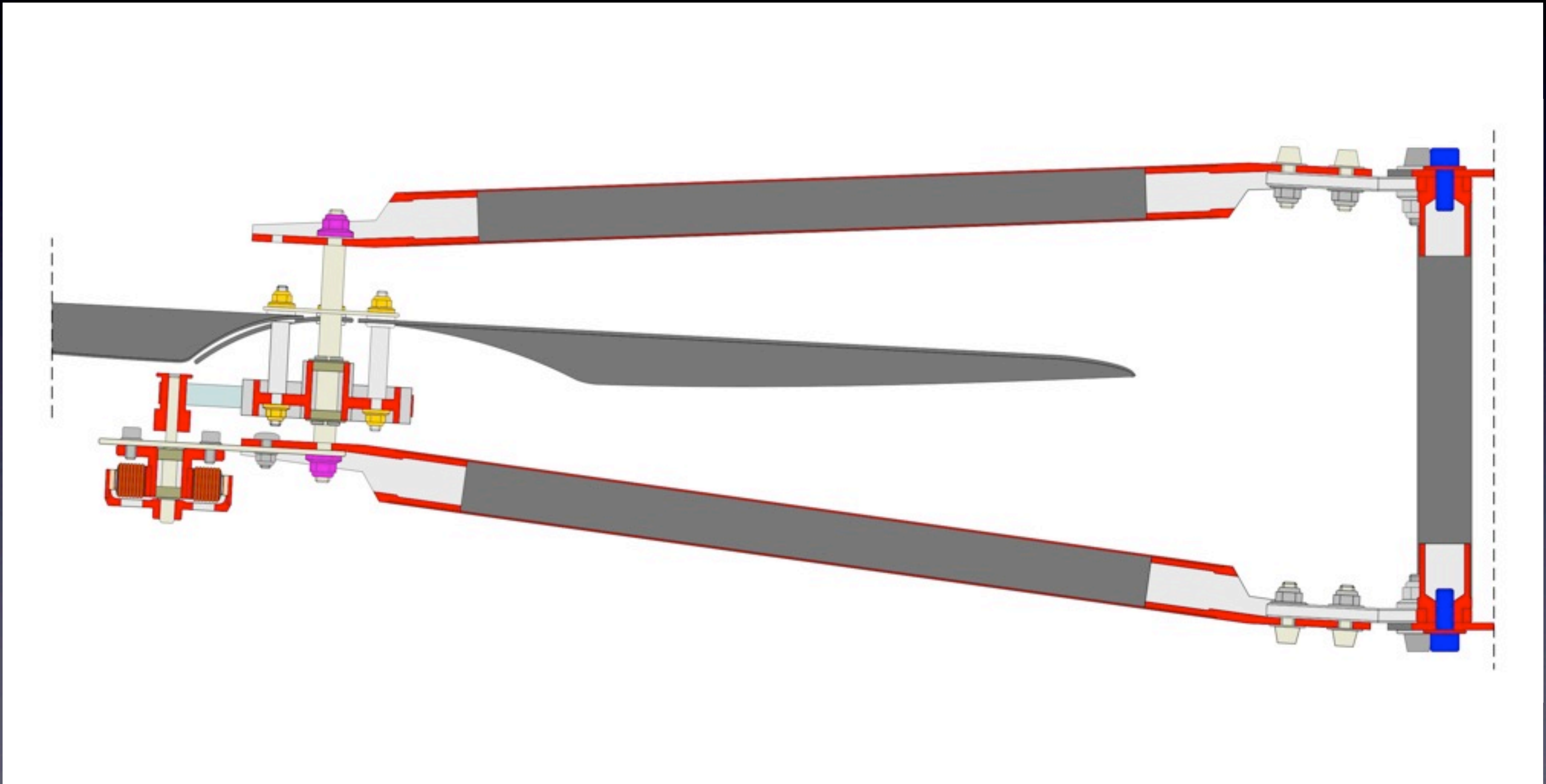
Open Source Blueprints



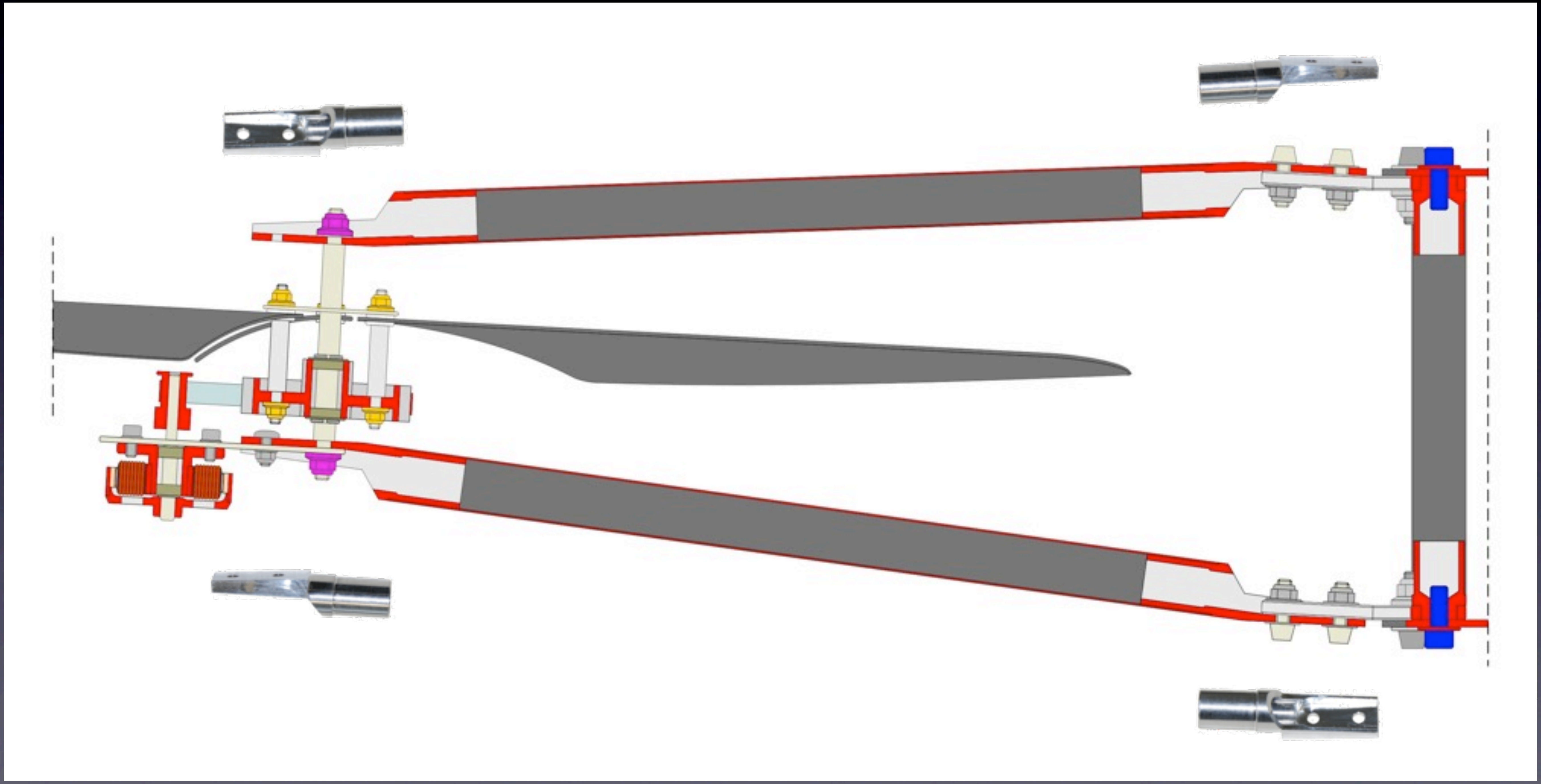
Minimal # of Different Parts



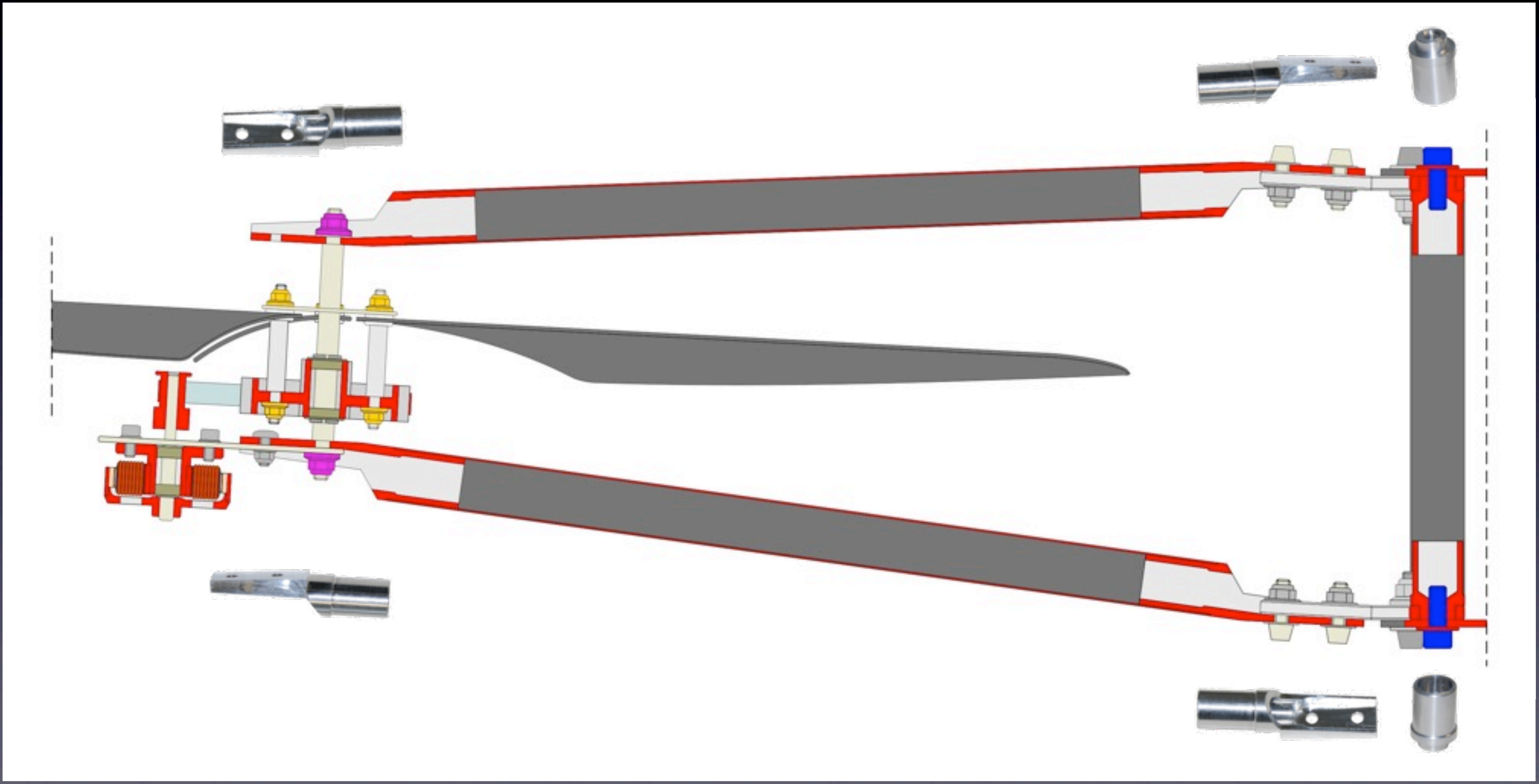
Minimal # of Different Parts



Minimal # of Different Parts

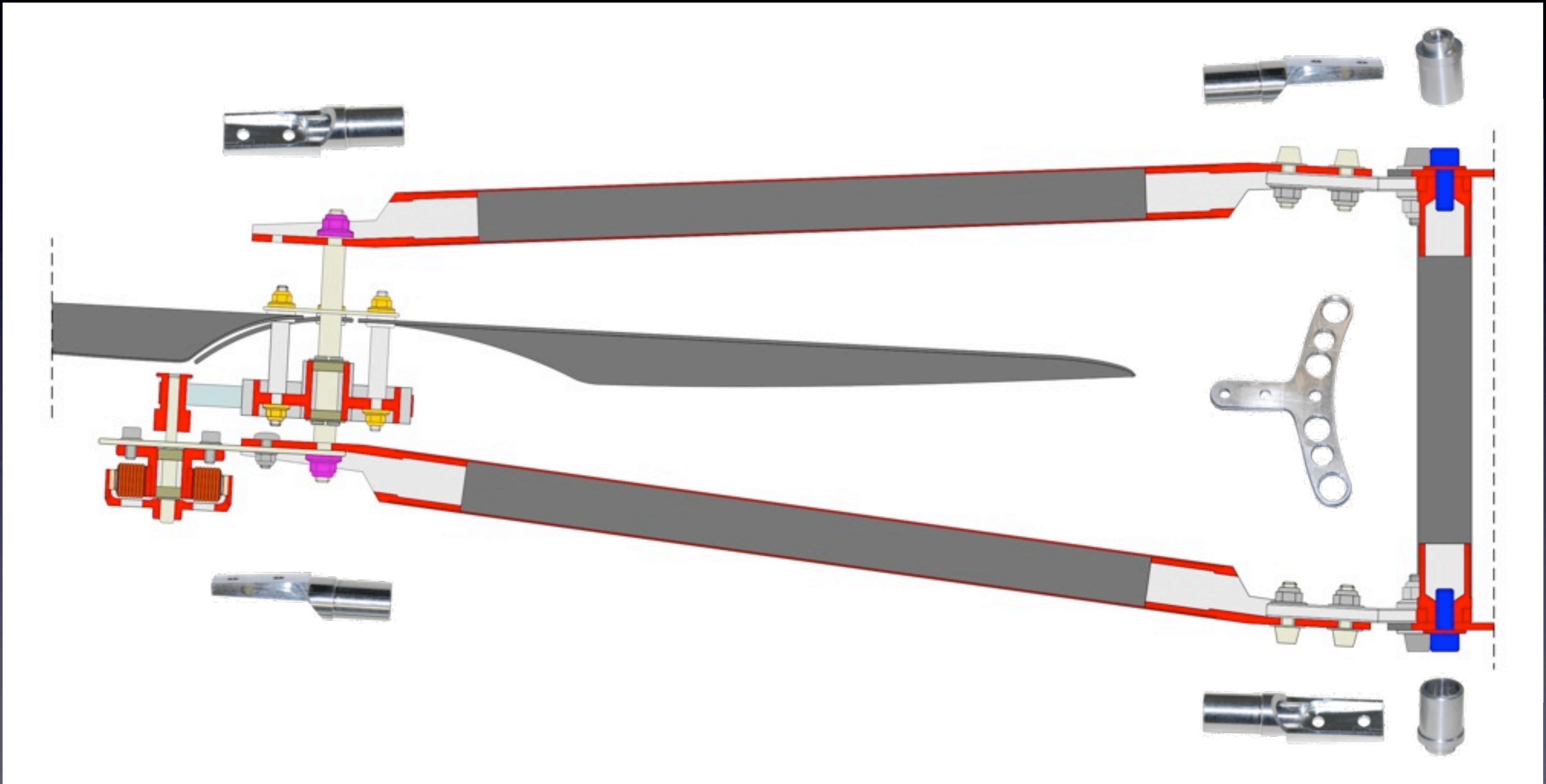


Minimal # of Different Parts



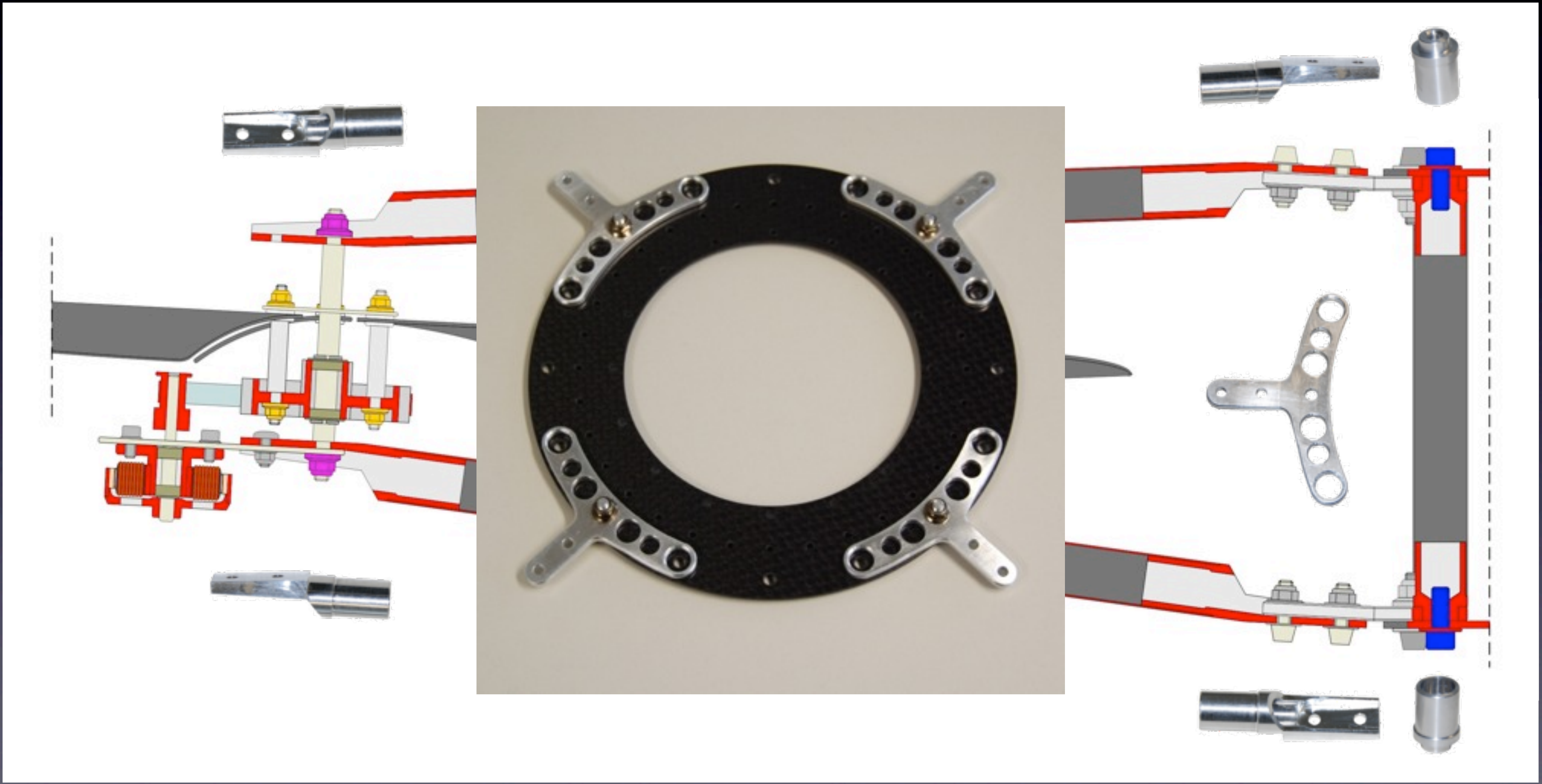
© C. Kirsch 2011

Minimal # of Different Parts

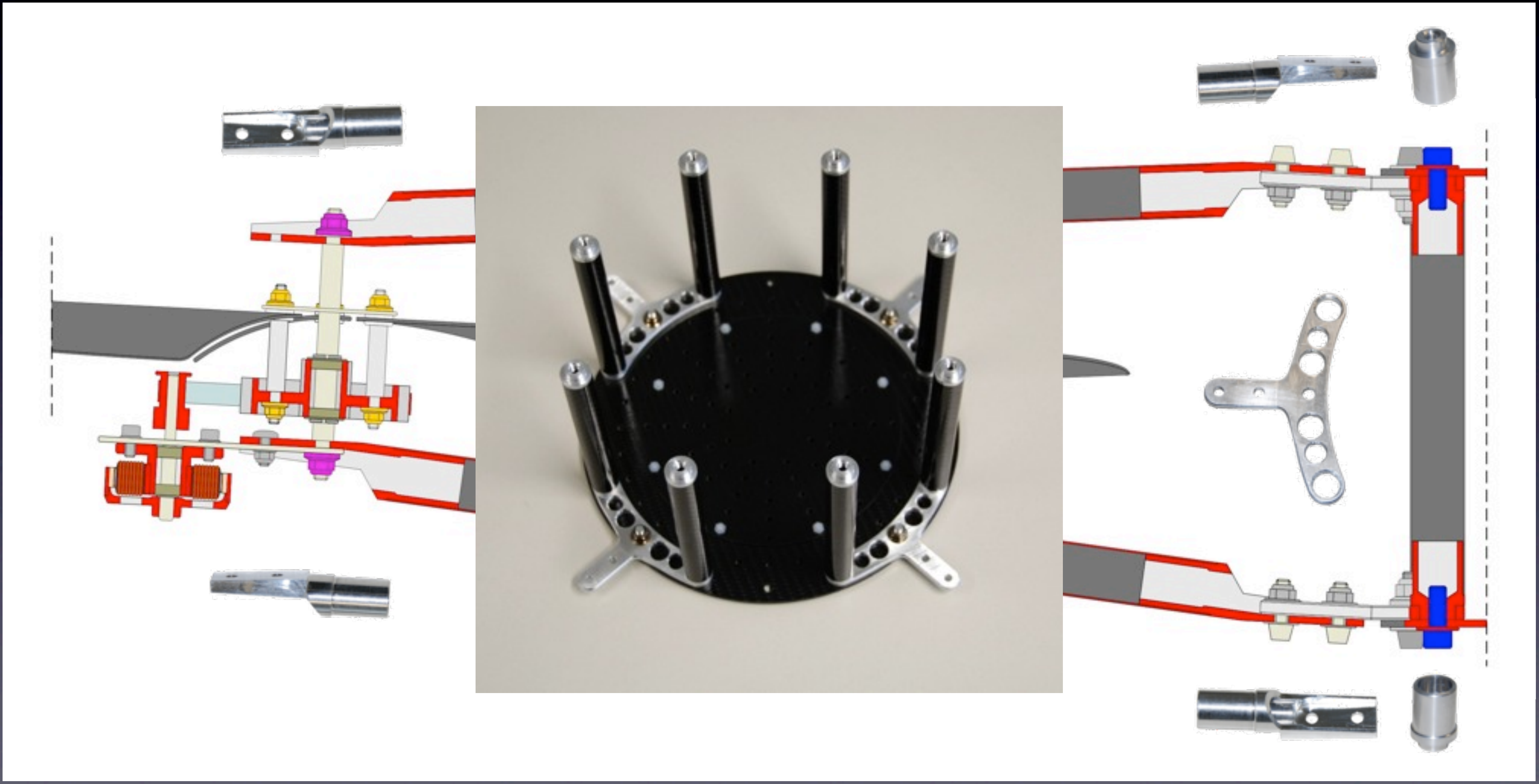


© C. Kirsch 2011

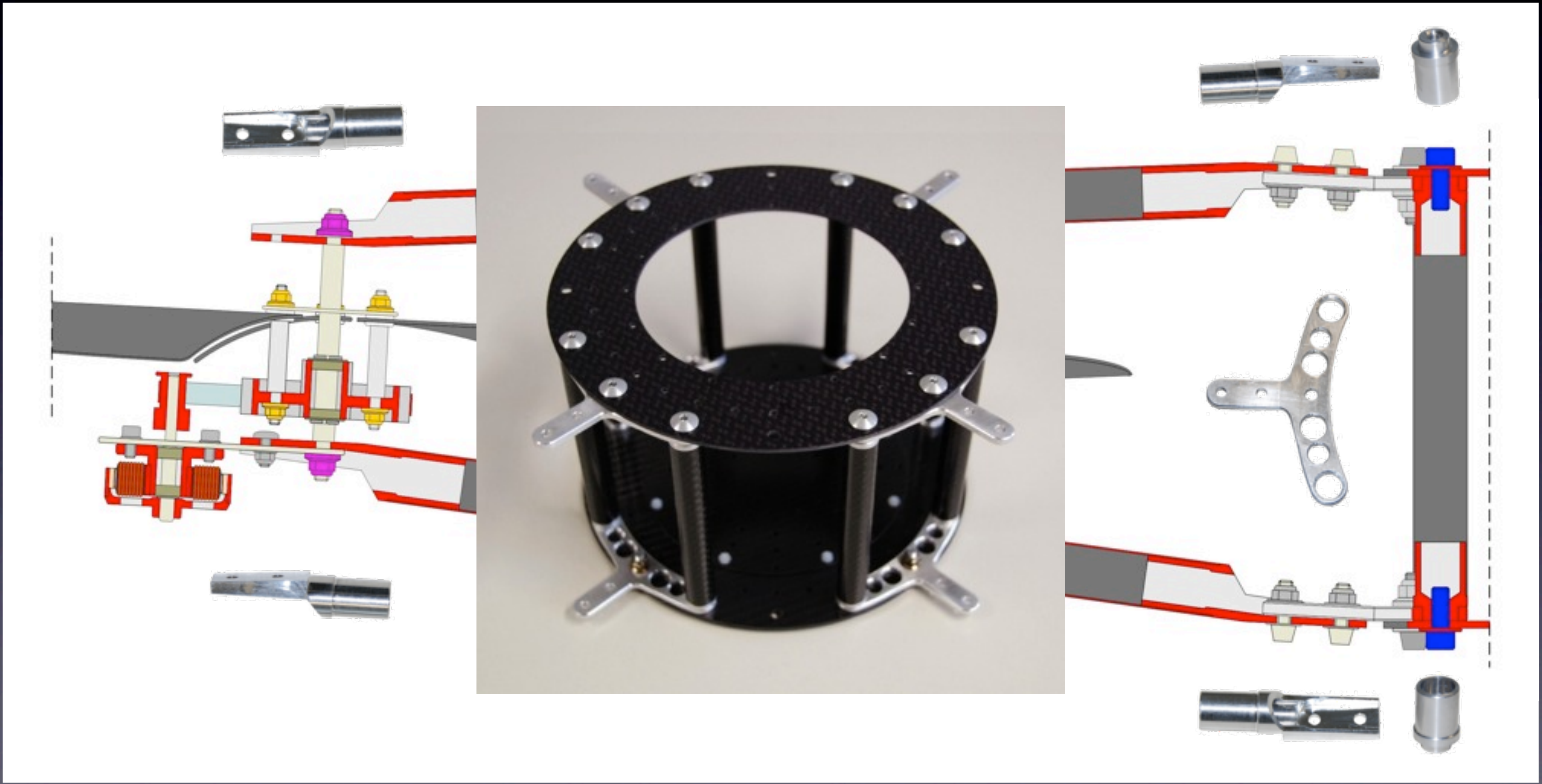
Minimal # of Different Parts



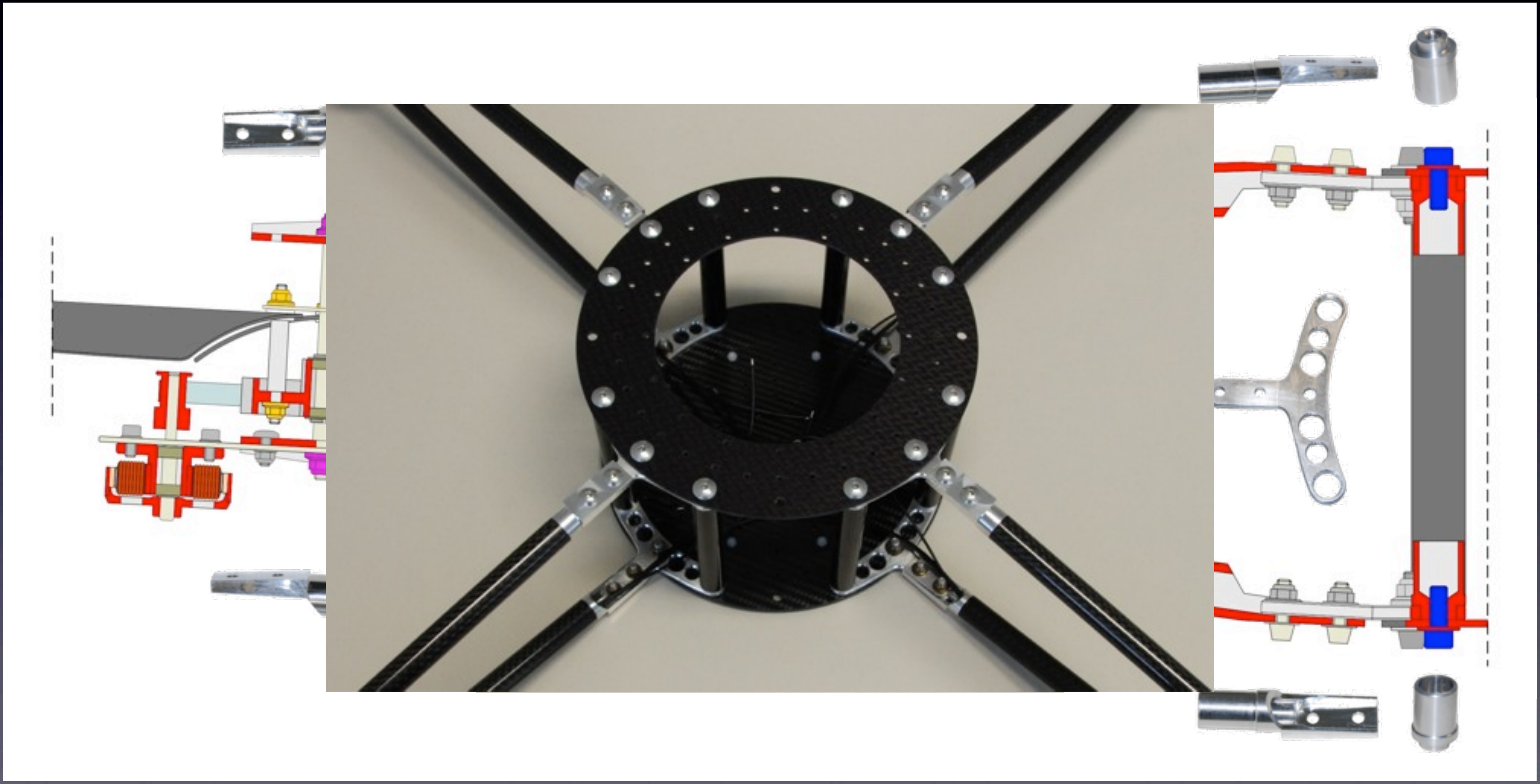
Minimal # of Different Parts



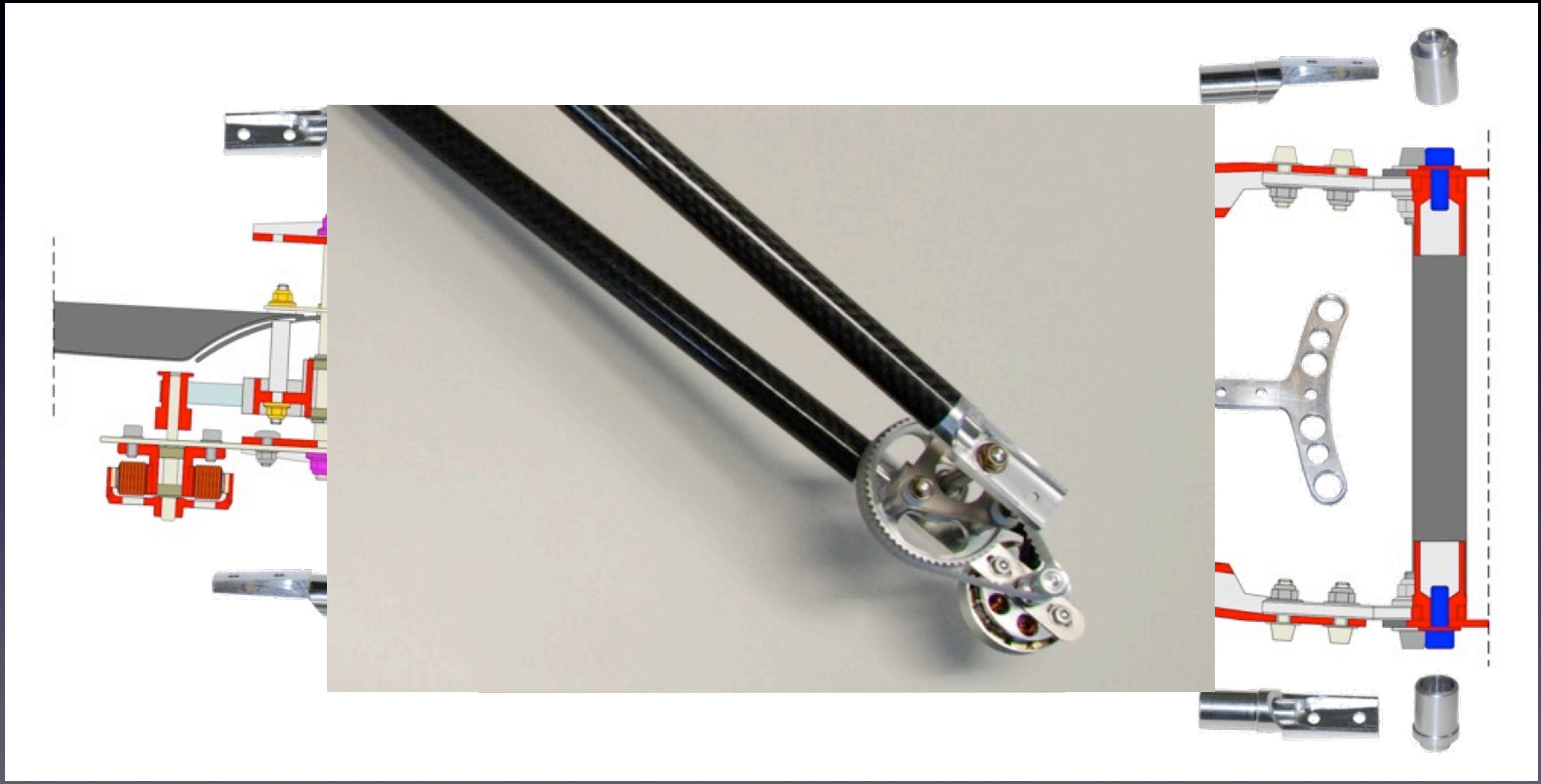
Minimal # of Different Parts

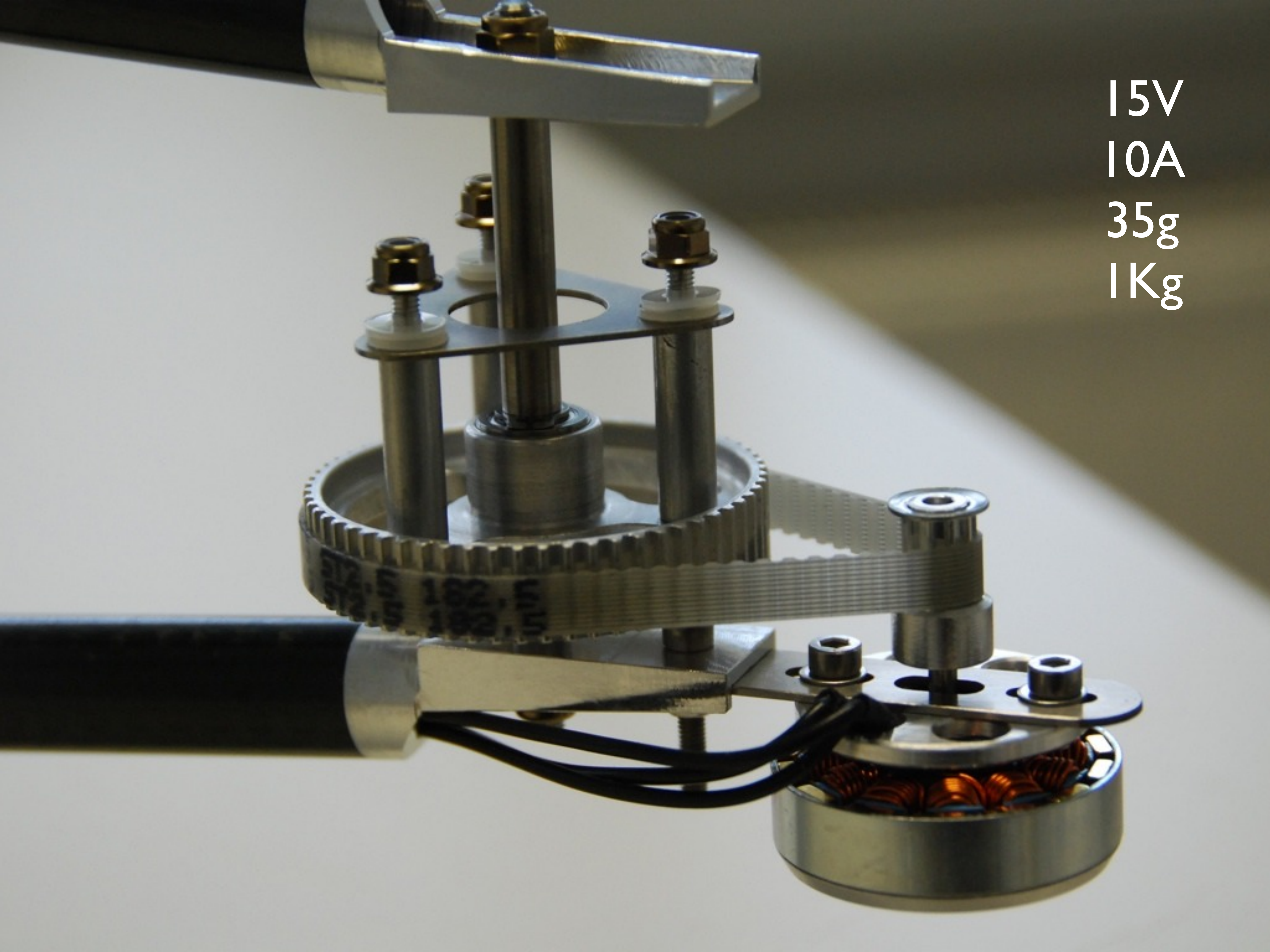


Minimal # of Different Parts



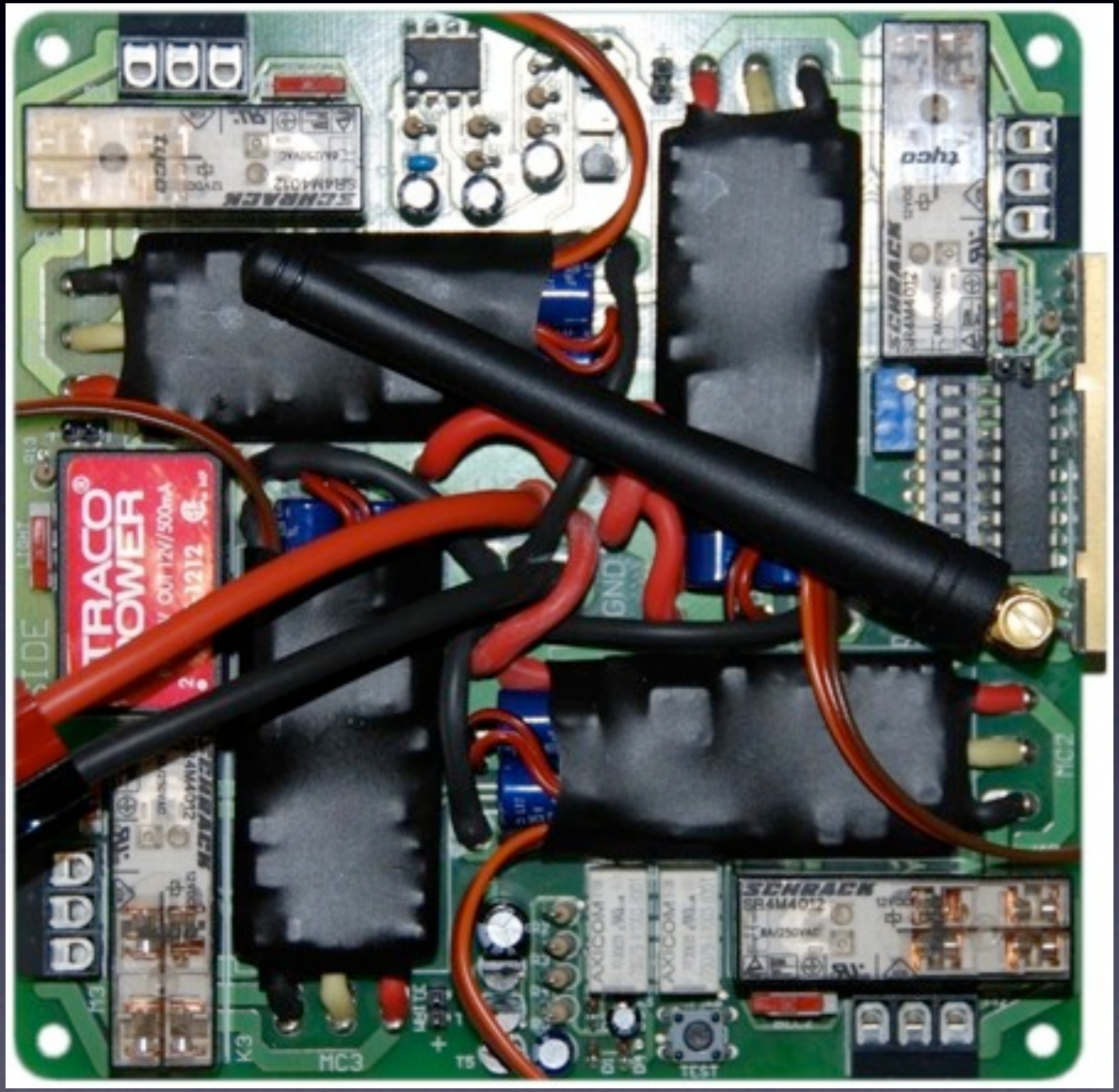
Minimal # of Different Parts





15V
10A
35g
1Kg

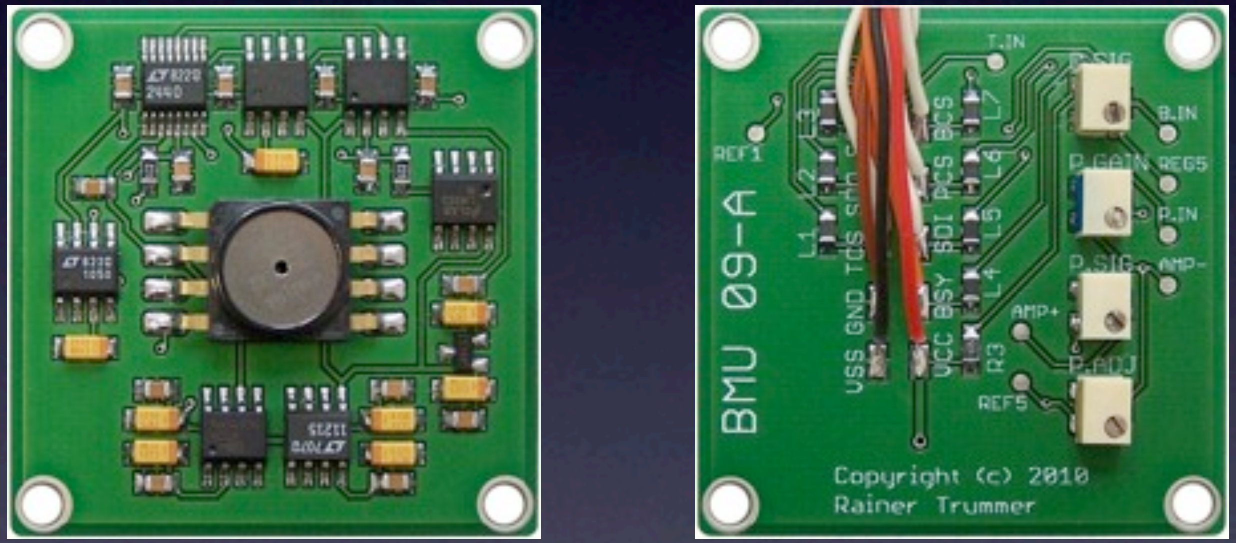
Custom Electronics



Power

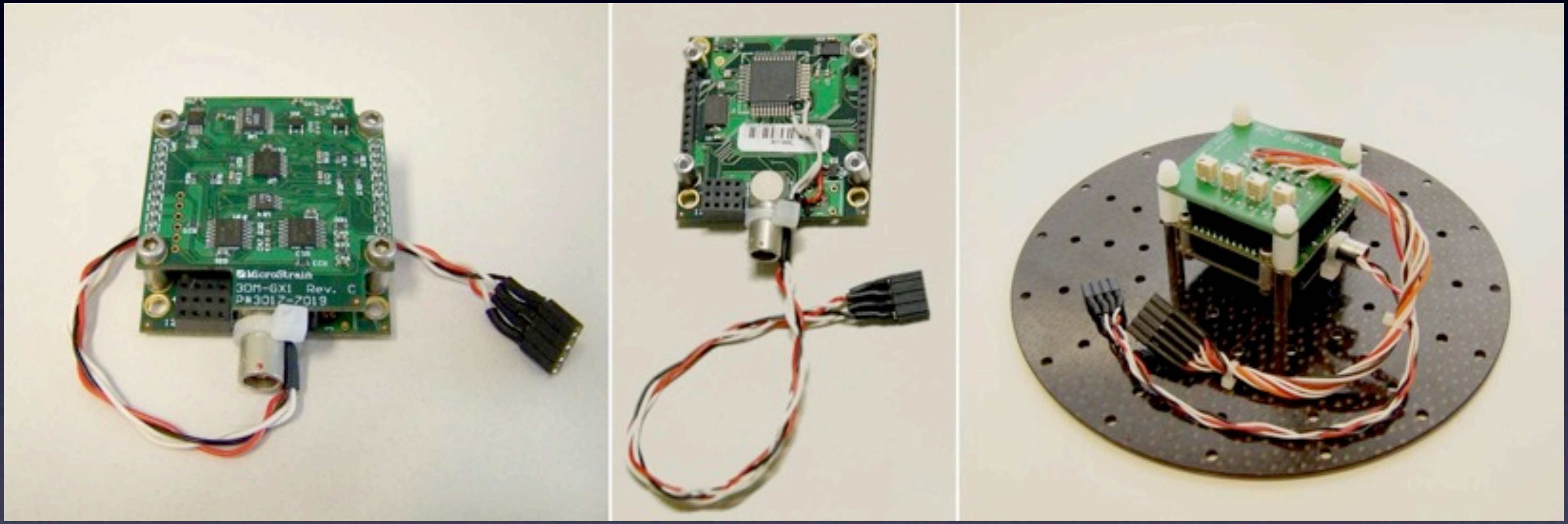
Remote

Custom Electronics



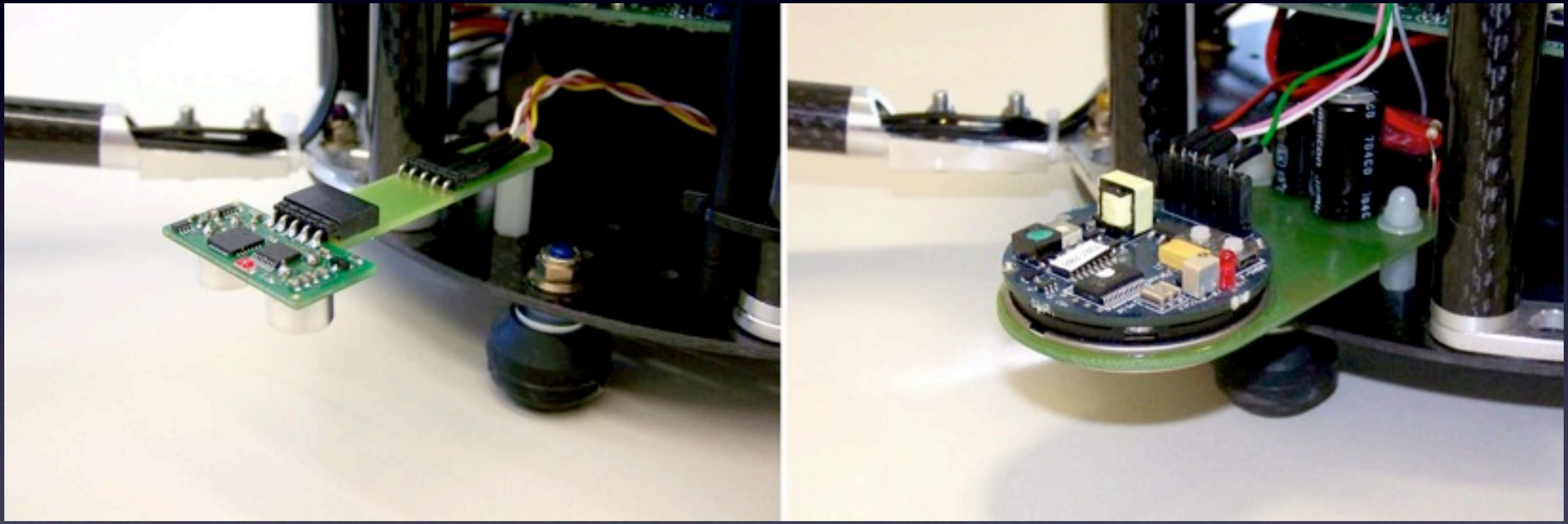
Barometer

Off-the-Shelf Stuff



Gyro

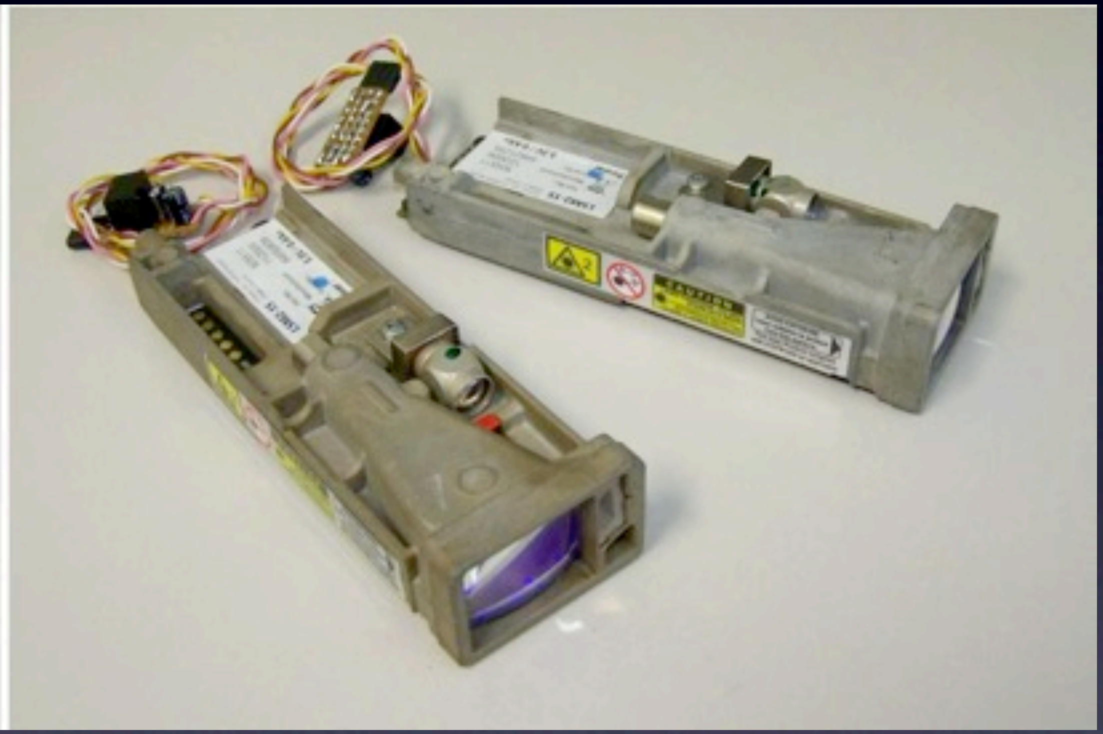
Off-the-Shelf Stuff



Ultrasonic

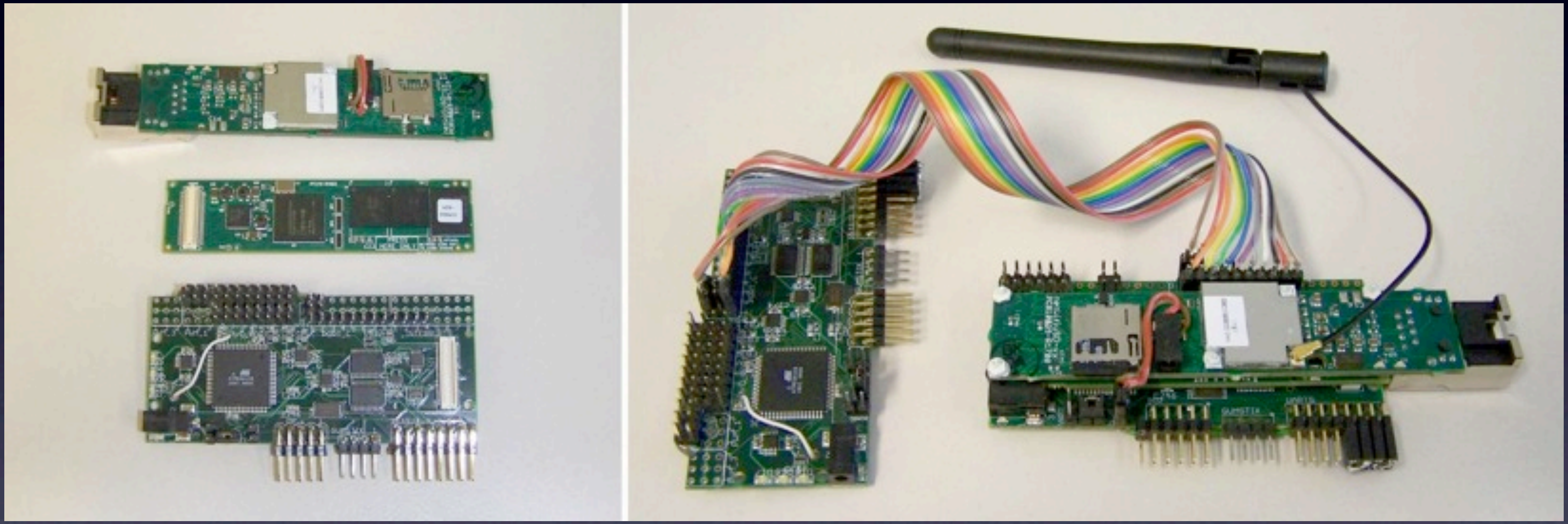
Off-the-Shelf Stuff

UWB RFID

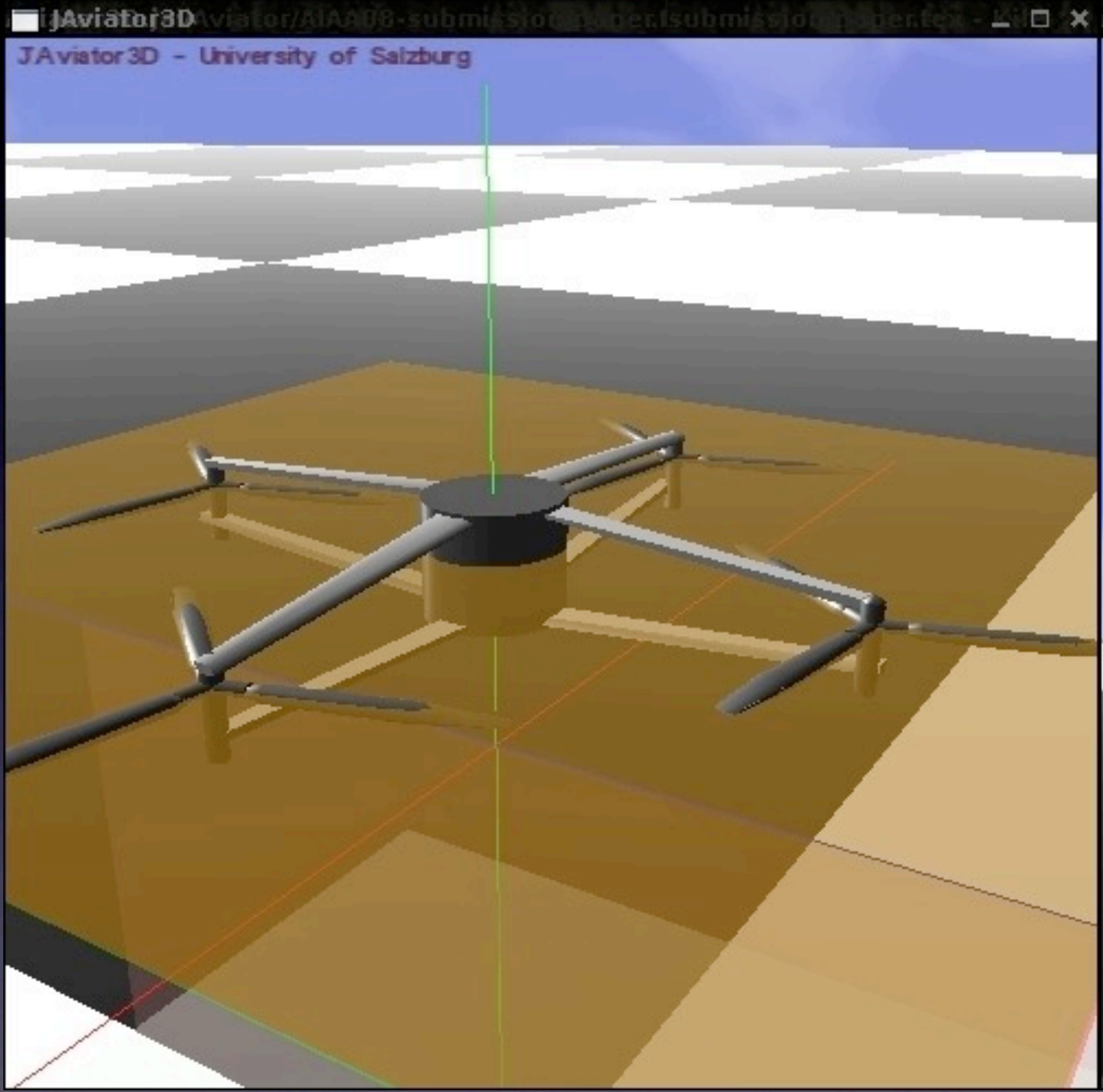


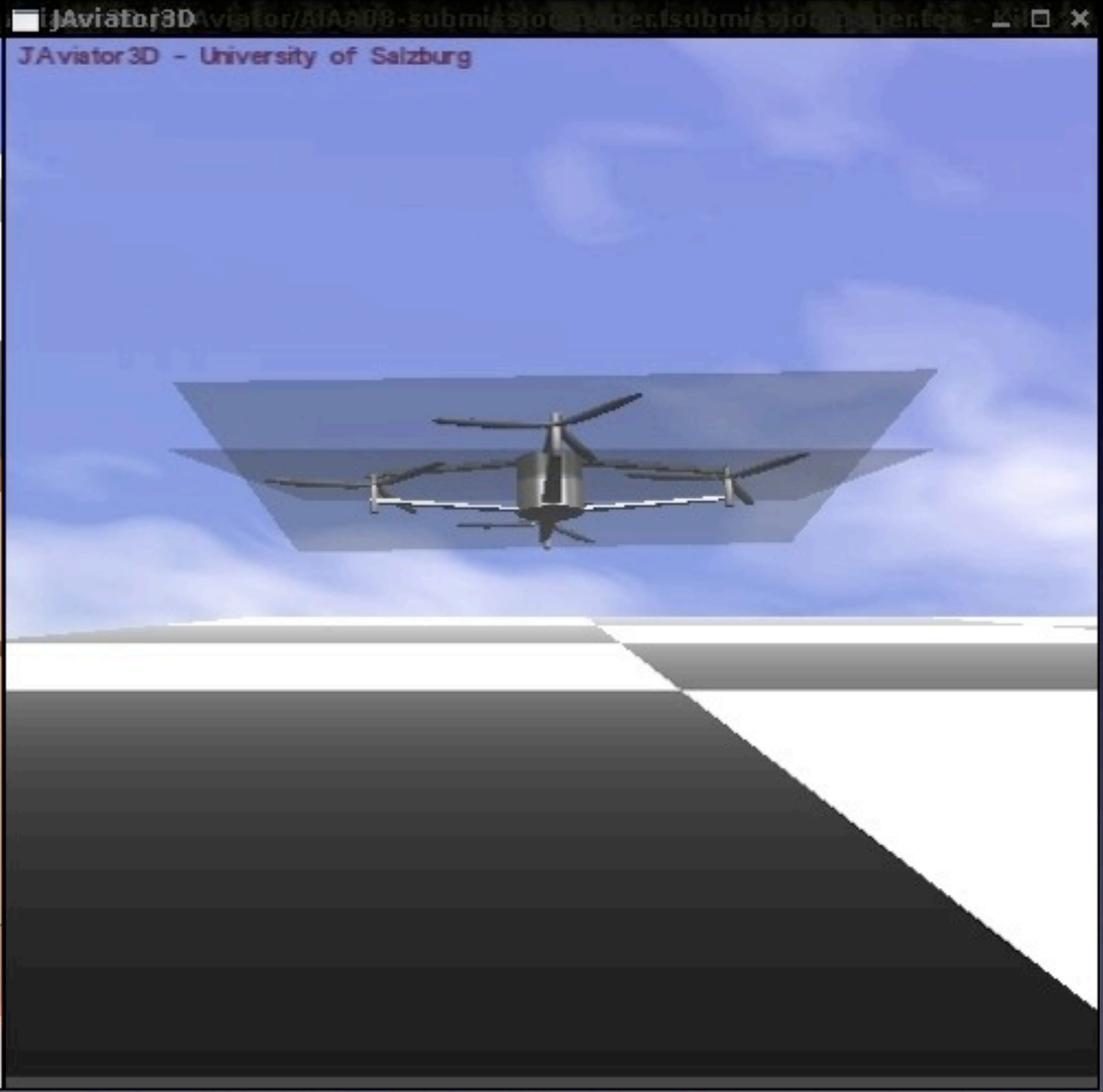
Laser

Off-the-Shelf Stuff



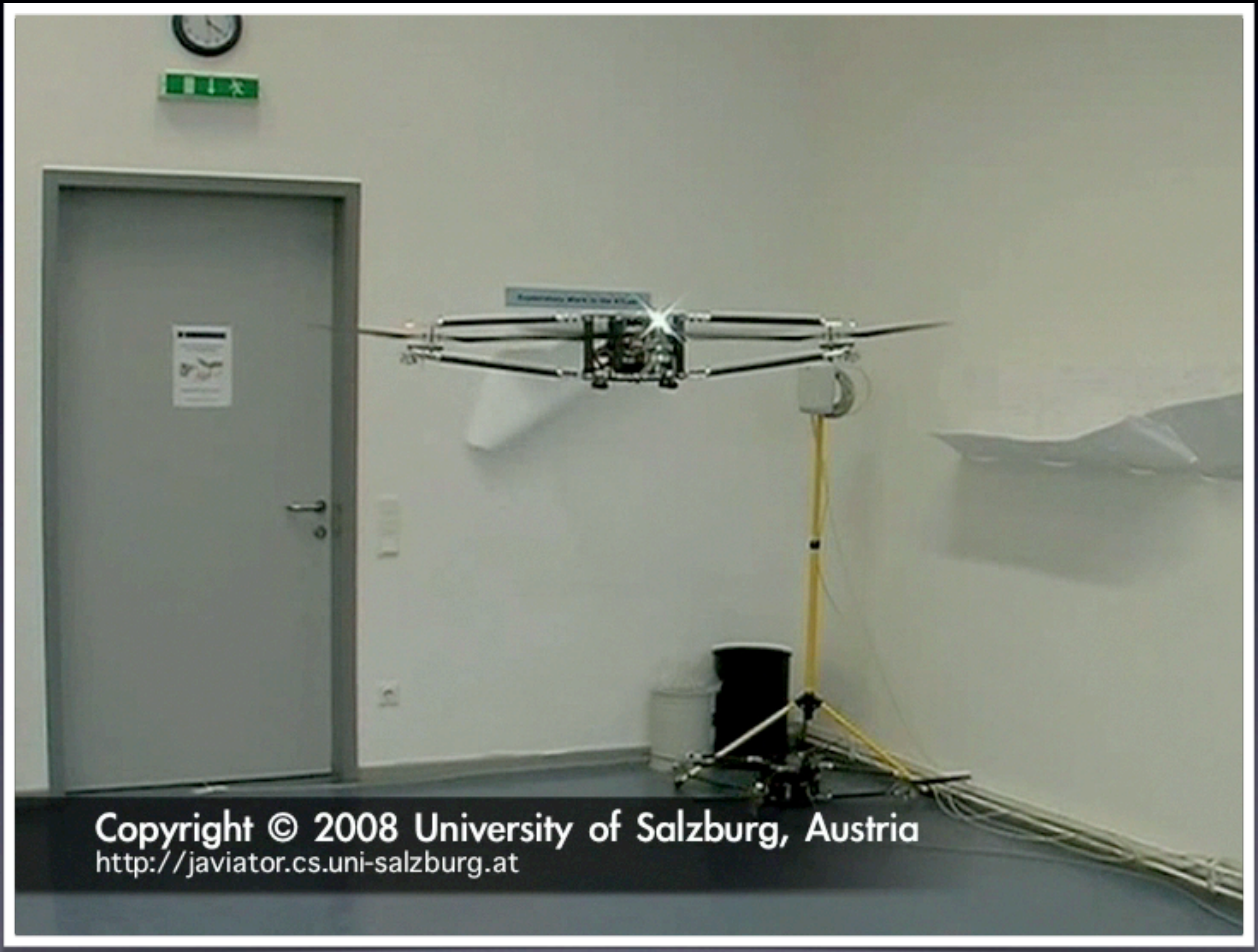
Gumstix


JAviator3D
JAviator3D - University of Salzburg


JAviator3D
JAviator3D - University of Salzburg

Indoor Flight
STARMAC Controller

Indoor Flight STARMAC Controller



Copyright © 2008 University of Salzburg, Austria
http://javiator.cs.uni-salzburg.at

Outdoor Flight
Salzburg Controller

Outdoor Flight
Salzburg Controller

Copyright © 2008 University of Salzburg, Austria
http://javiator.cs.uni-salzburg.at

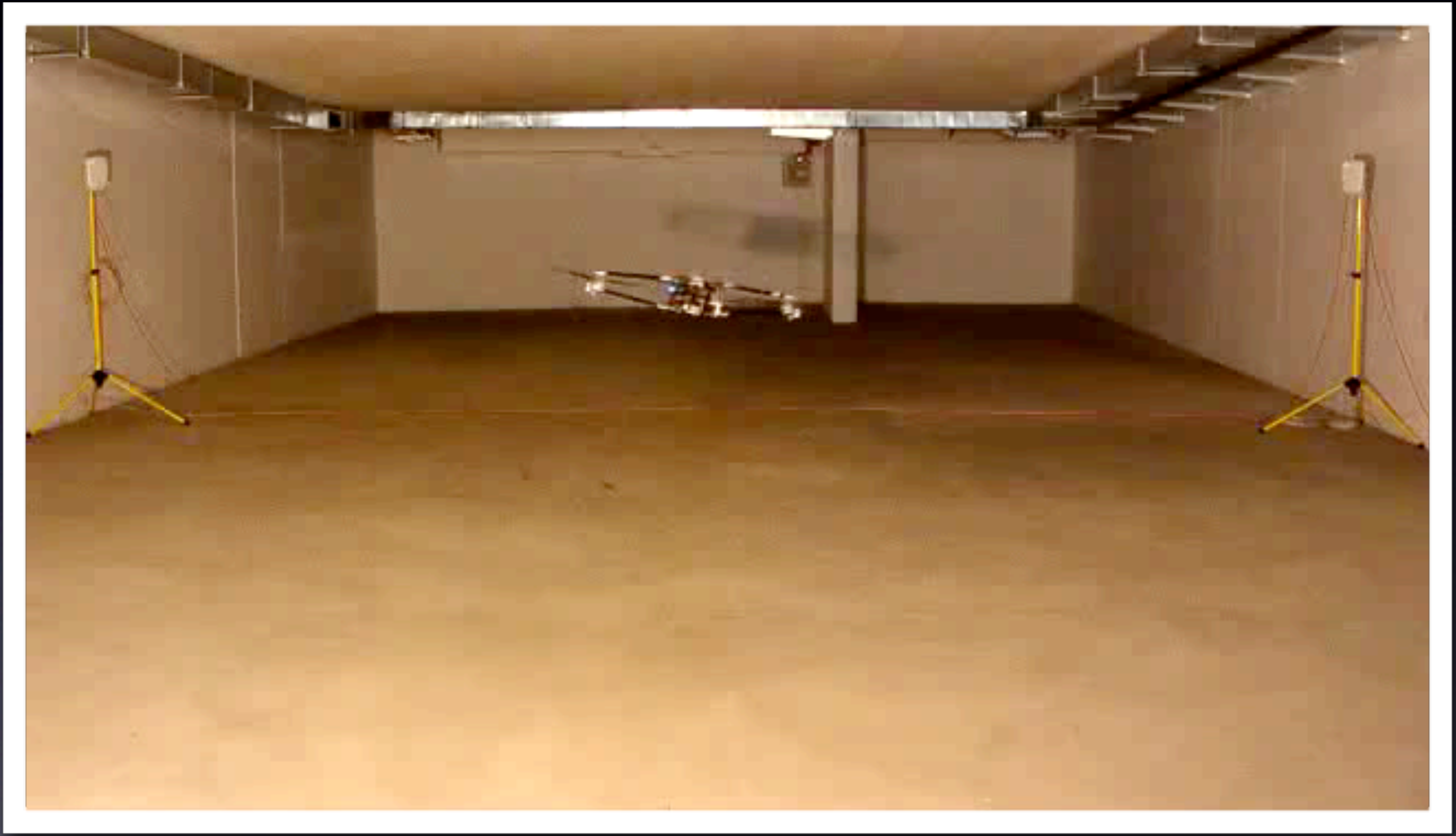
More Recent: Yawing

More Recent: Yawing

© C. Kirsch 2011

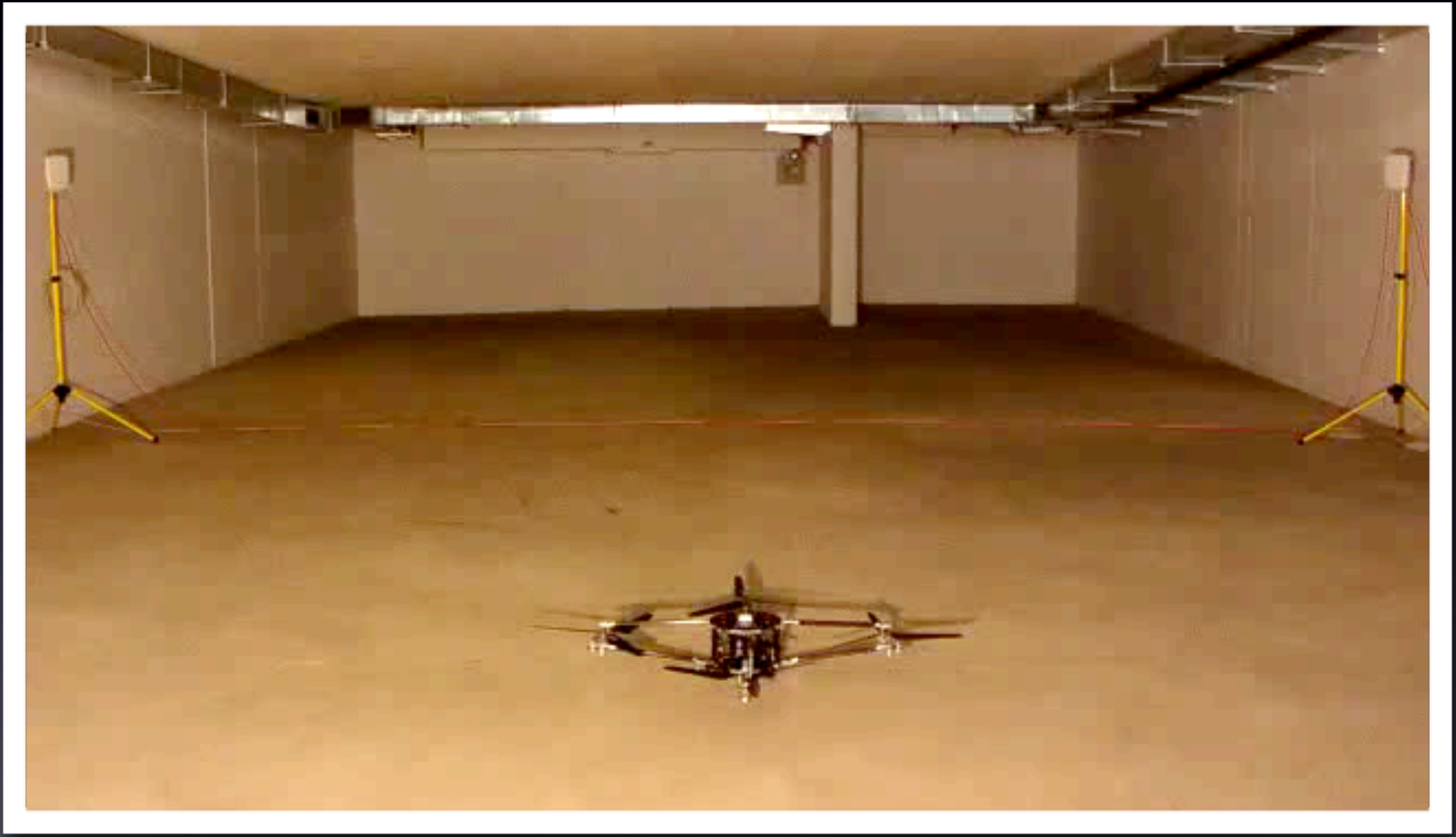
Oops

Oops



Autonomous

Autonomous



A Mobile Server

- IP address
- location

A Mobile Server

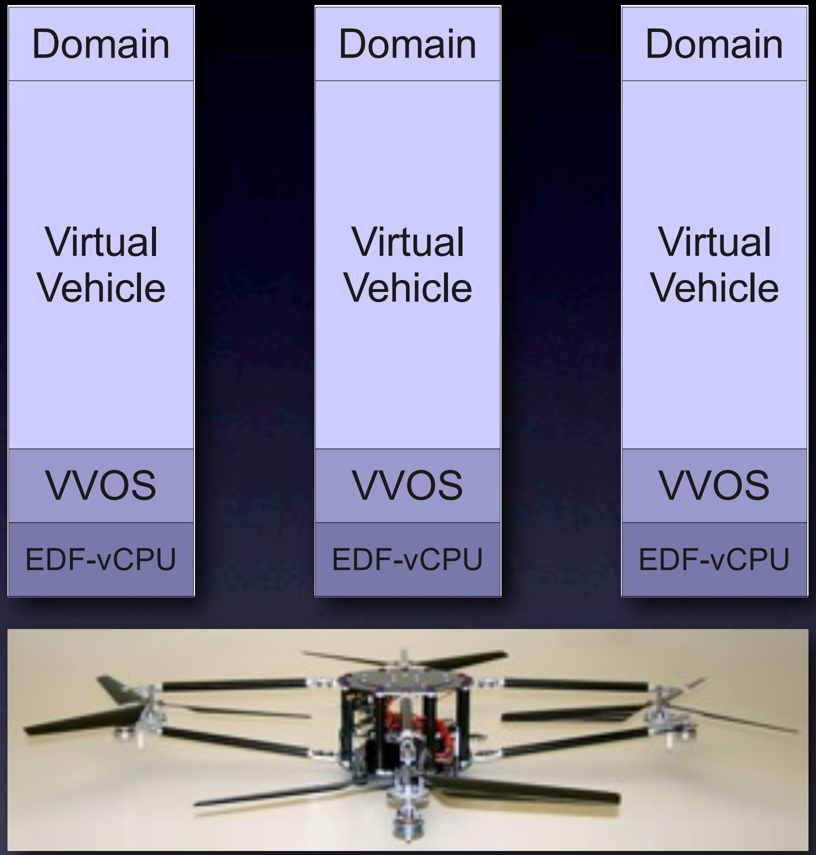
- IP address
- location
- capabilities

A Mobile Server

- IP address
- location
- capabilities
- motion

A Mobile Server

- IP address
- location
- capabilities
- motion

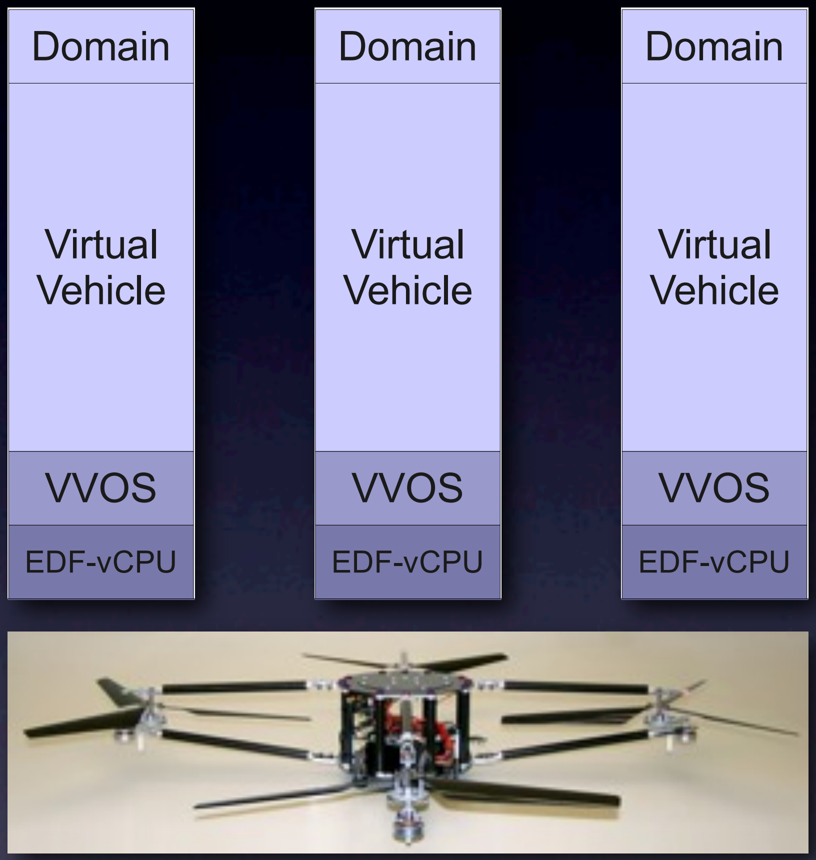


- IP address
- location
- capabilities
- motion

- IP address
- location
- capabilities
- motion

A Mobile Server

- IP address
- location
- capabilities
- motion



Domain | Domain | Domain

Virtual Vehicle | Virtual Vehicle | Virtual Vehicle

VVOS | VVOS | VVOS

EDF-vCPU | EDF-vCPU | EDF-vCPU

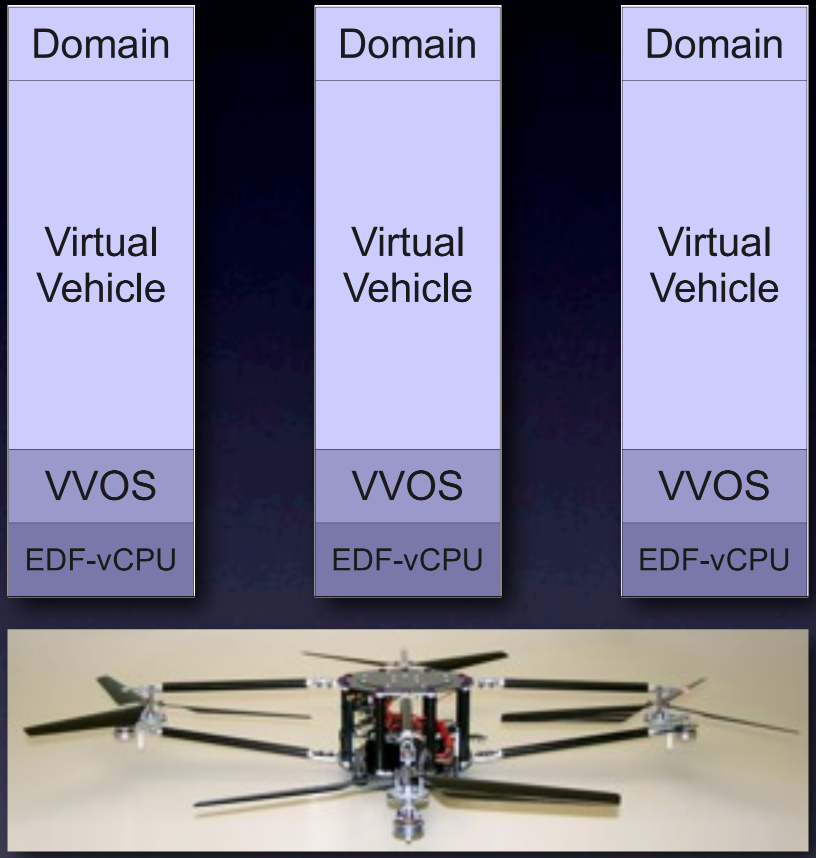
- IP address
- location
- capabilities
- motion

restricted

- IP address
- location
- capabilities
- motion

A Mobile Server

- IP address
- location
- capabilities
- motion



Domain	Domain	Domain
Virtual Vehicle	Virtual Vehicle	Virtual Vehicle
VVOS	VVOS	VVOS
EDF-vCPU	EDF-vCPU	EDF-vCPU

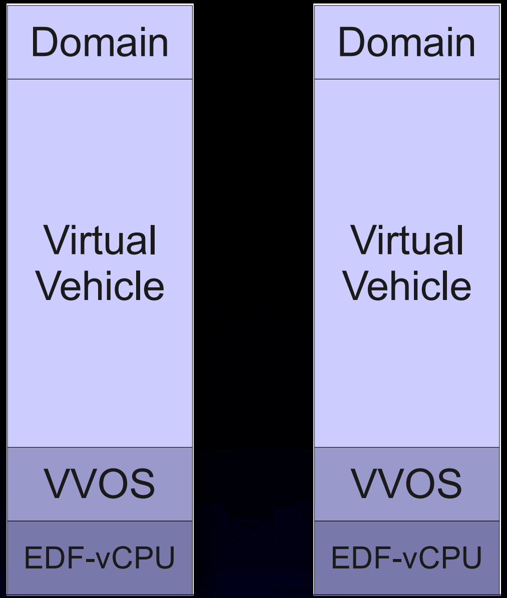
- IP address
- location
- capabilities
- motion

restricted

idealized

- IP address
- location
- capabilities
- motion

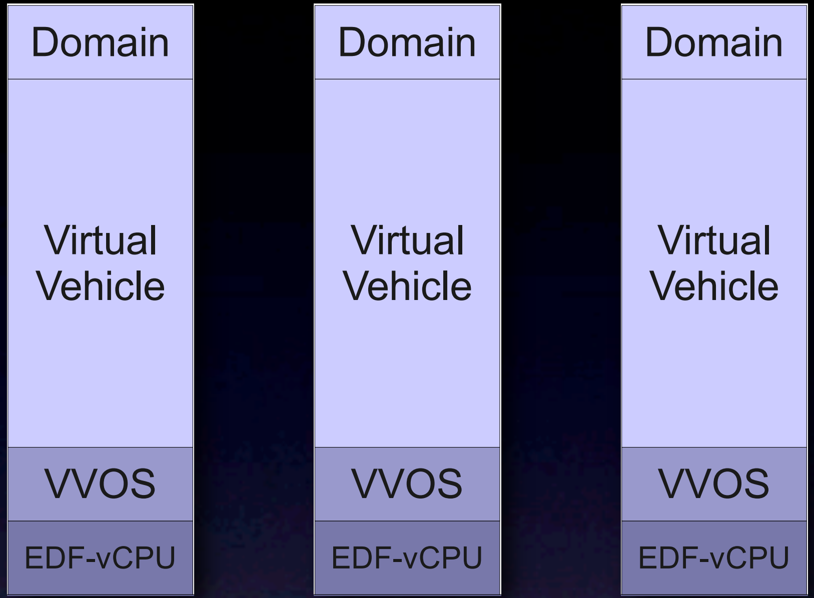
A Cyber-Physical Cloud [HotCloud 2010]


Domain | Domain

Virtual Vehicle | Virtual Vehicle

VVOS | VVOS

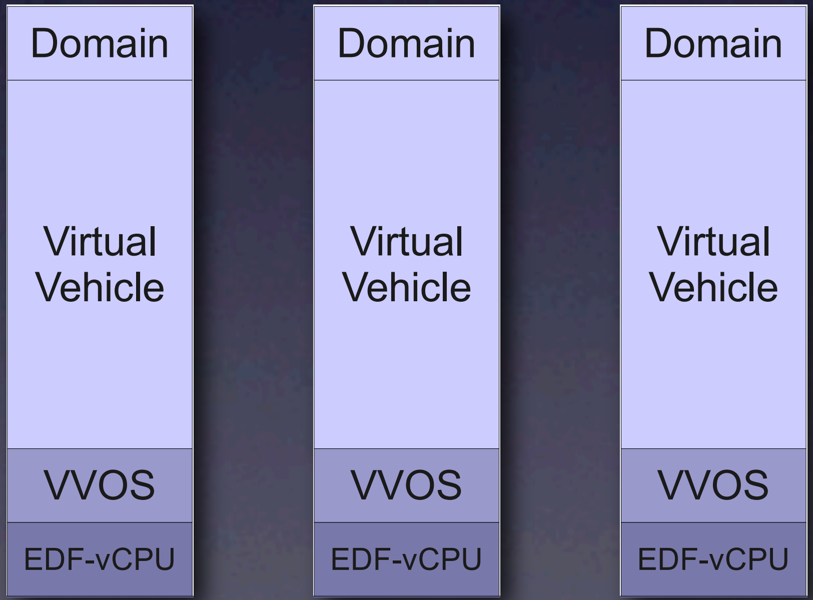
EDF-vCPU | EDF-vCPU


Domain | Domain | Domain

Virtual Vehicle | Virtual Vehicle | Virtual Vehicle

VVOS | VVOS | VVOS

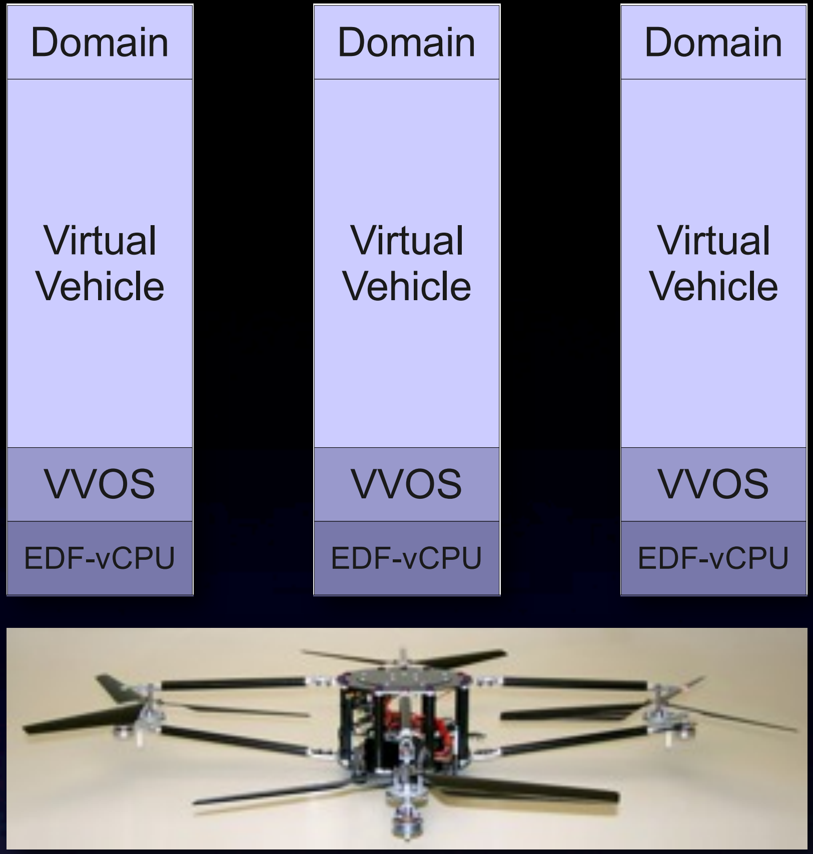
EDF-vCPU | EDF-vCPU | EDF-vCPU


Domain | Domain | Domain

Virtual Vehicle | Virtual Vehicle | Virtual Vehicle

VVOS | VVOS | VVOS

EDF-vCPU | EDF-vCPU | EDF-vCPU



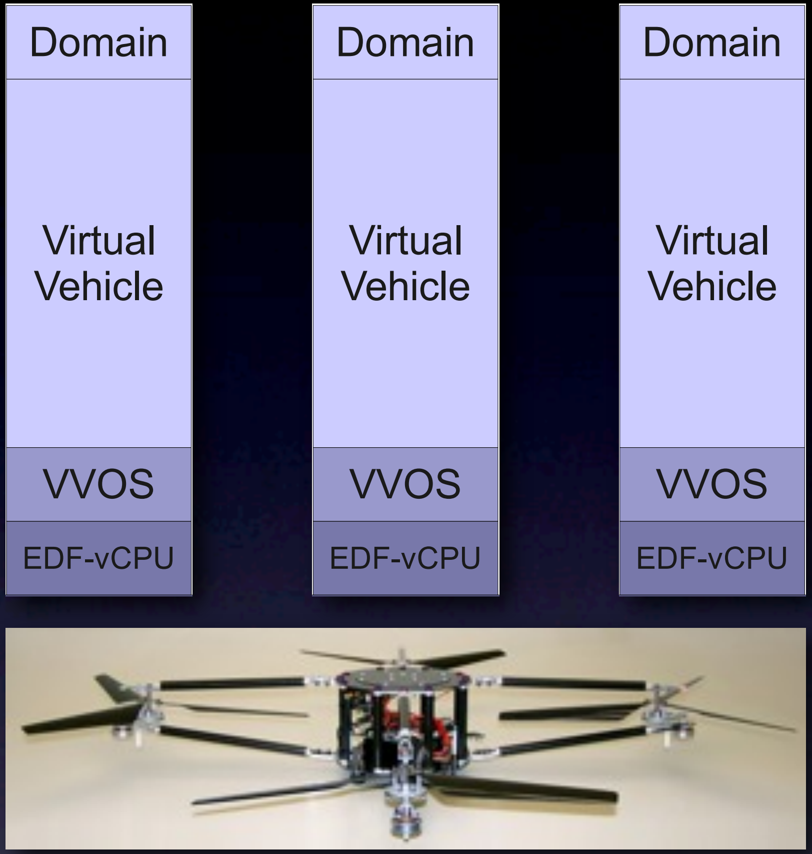
Domain | Domain | Domain

Virtual Vehicle | Virtual Vehicle | Virtual Vehicle

VVOS | VVOS | VVOS

EDF-vCPU | EDF-vCPU | EDF-vCPU

migration
=
flying



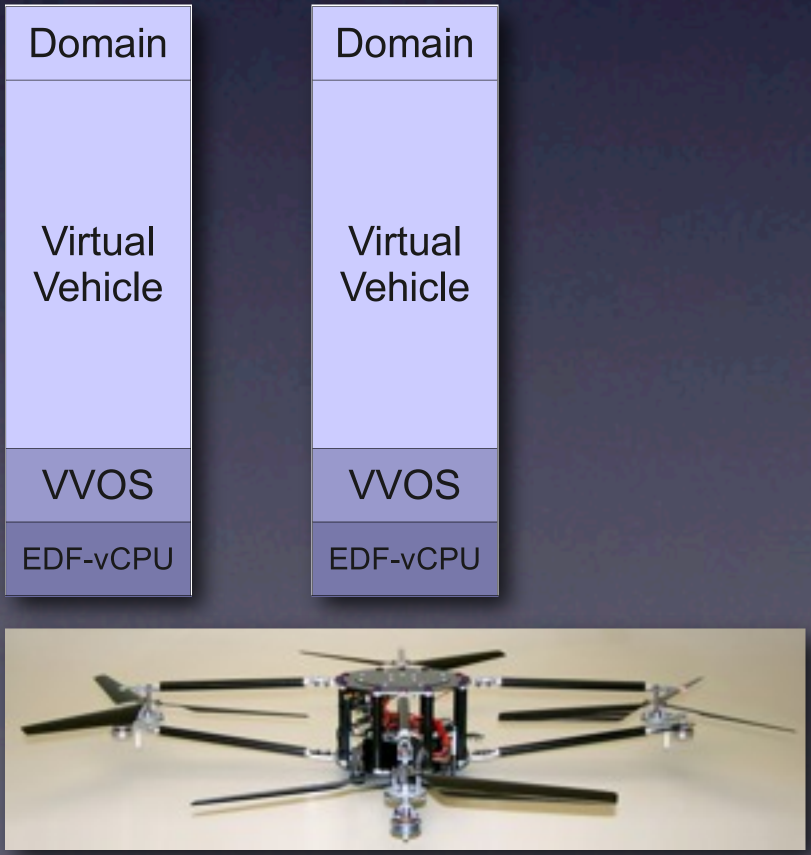
Domain | Domain | Domain

Virtual Vehicle | Virtual Vehicle | Virtual Vehicle

VVOS | VVOS | VVOS

EDF-vCPU | EDF-vCPU | EDF-vCPU

A Cyber-Physical Cloud [HotCloud 2010]



Domain | Domain

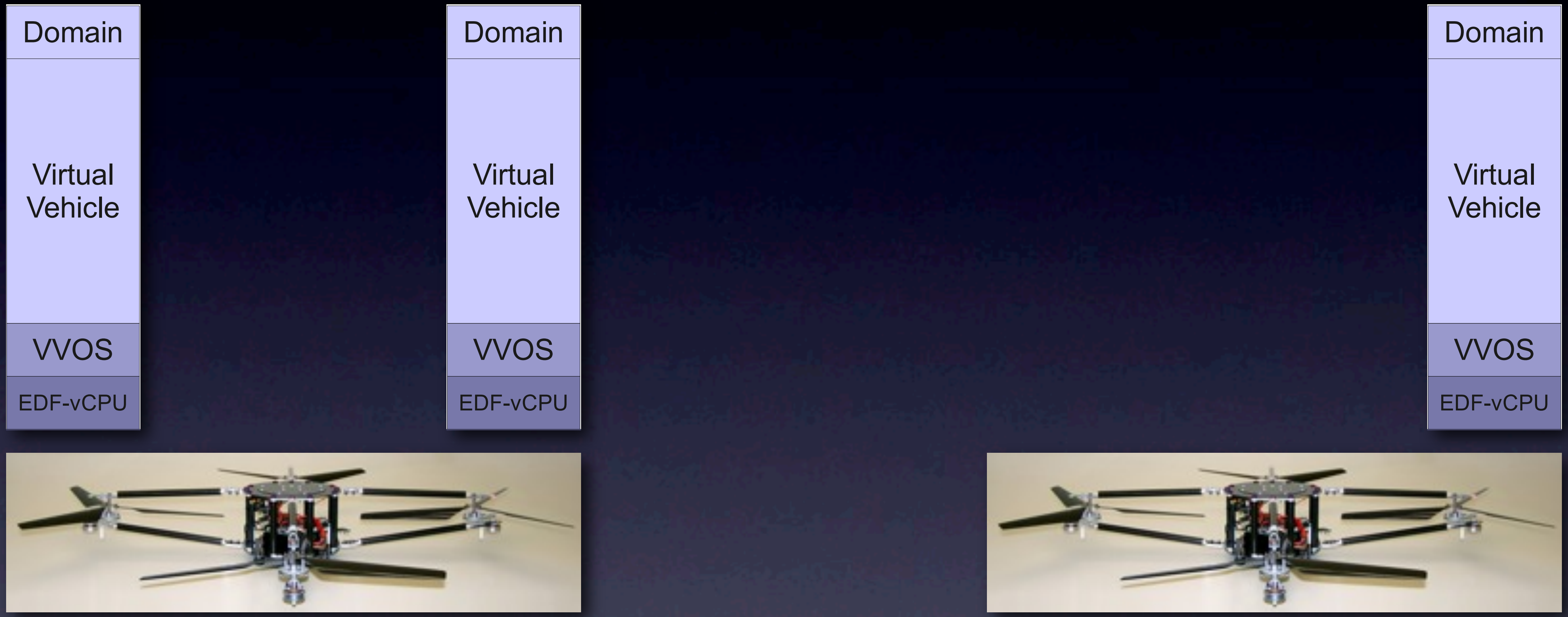
Virtual Vehicle | Virtual Vehicle

VVOS | VVOS

EDF-vCPU | EDF-vCPU

Virtual Vehicle Demo

by Florian Landolt and Andreas Rottmann



Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

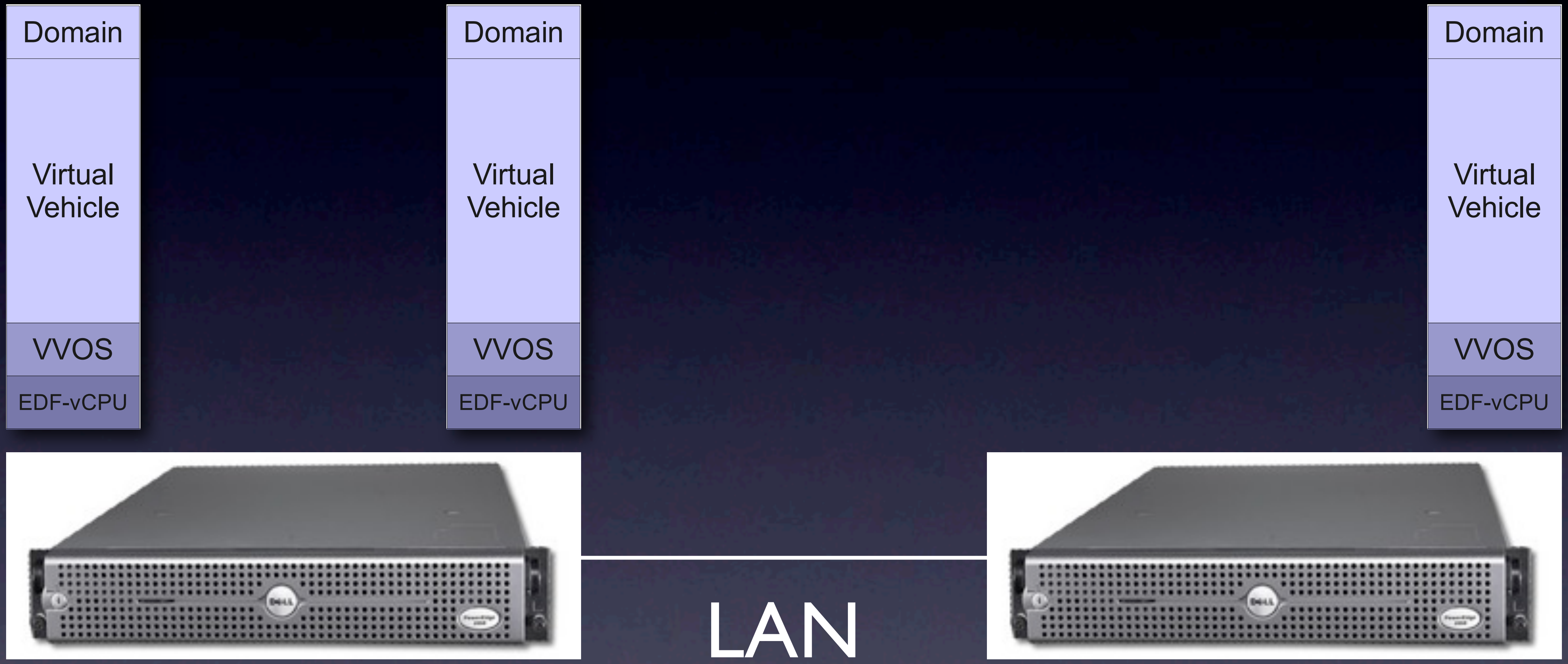
Virtual Vehicle

VVOS

EDF-vCPU

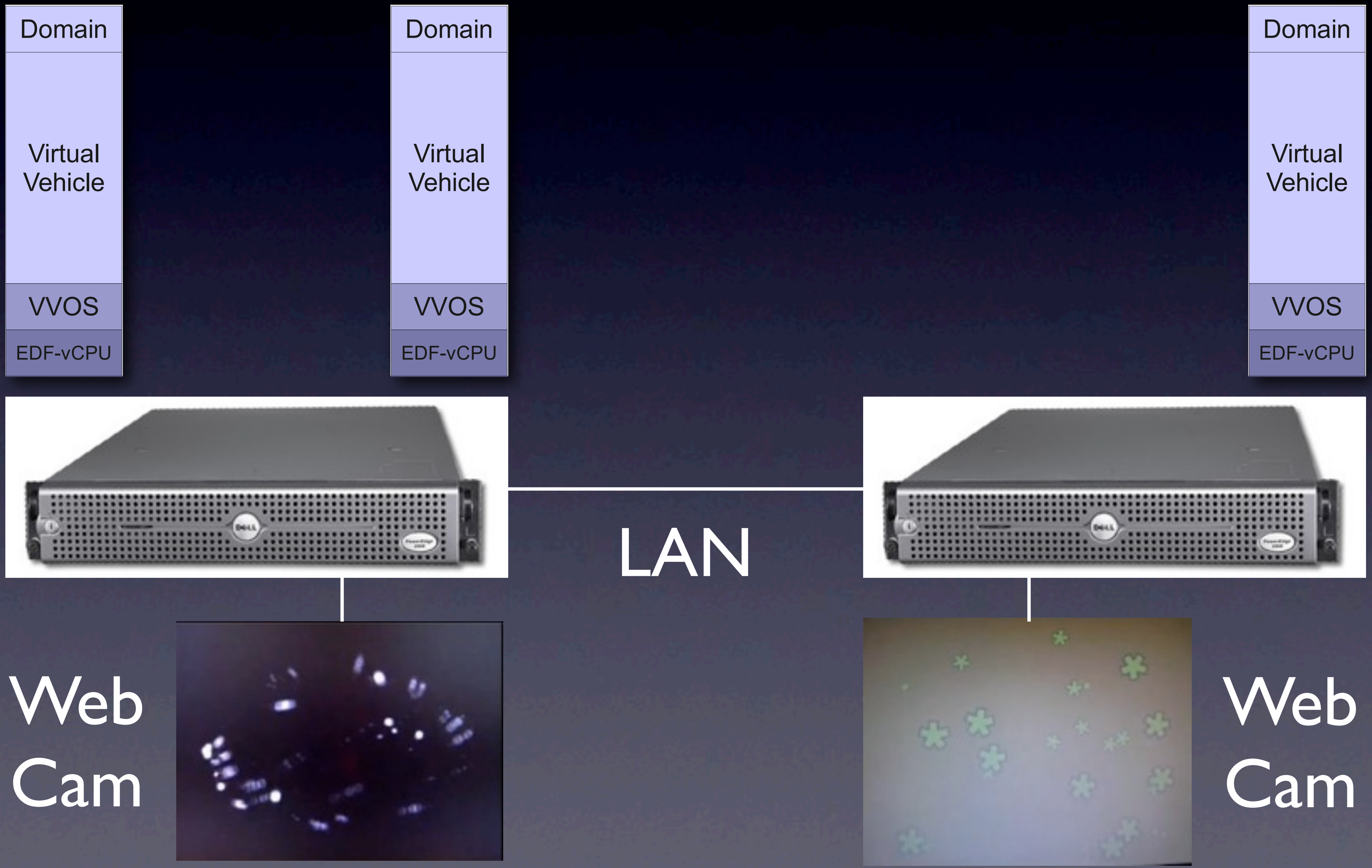
Virtual Vehicle Demo

by Florian Landolt and Andreas Rottmann



Virtual Vehicle Demo

by Florian Landolt and Andreas Rottmann



Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

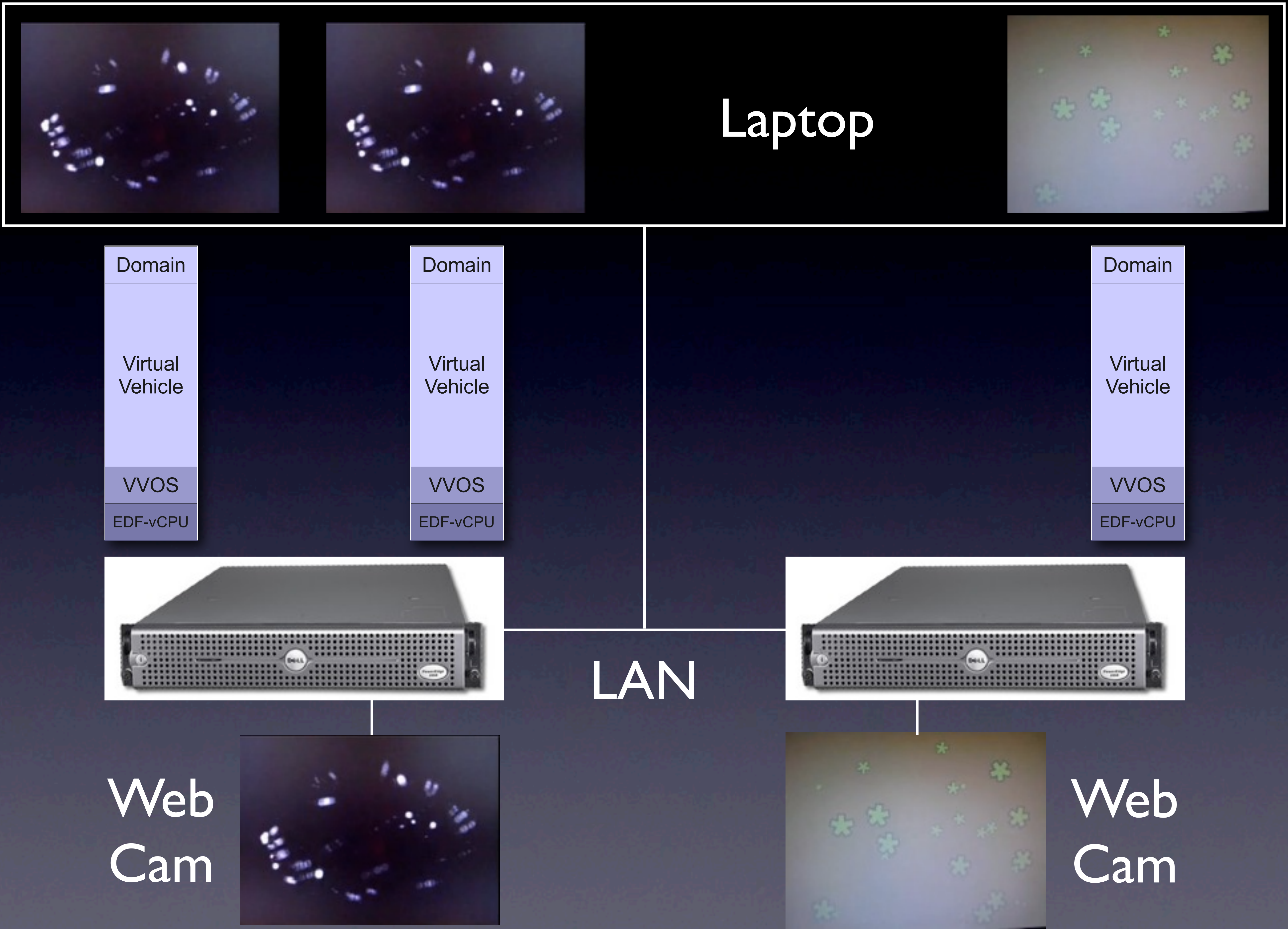
VVOS

EDF-vCPU

LAN

Web Cam

Web Cam



Laptop

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

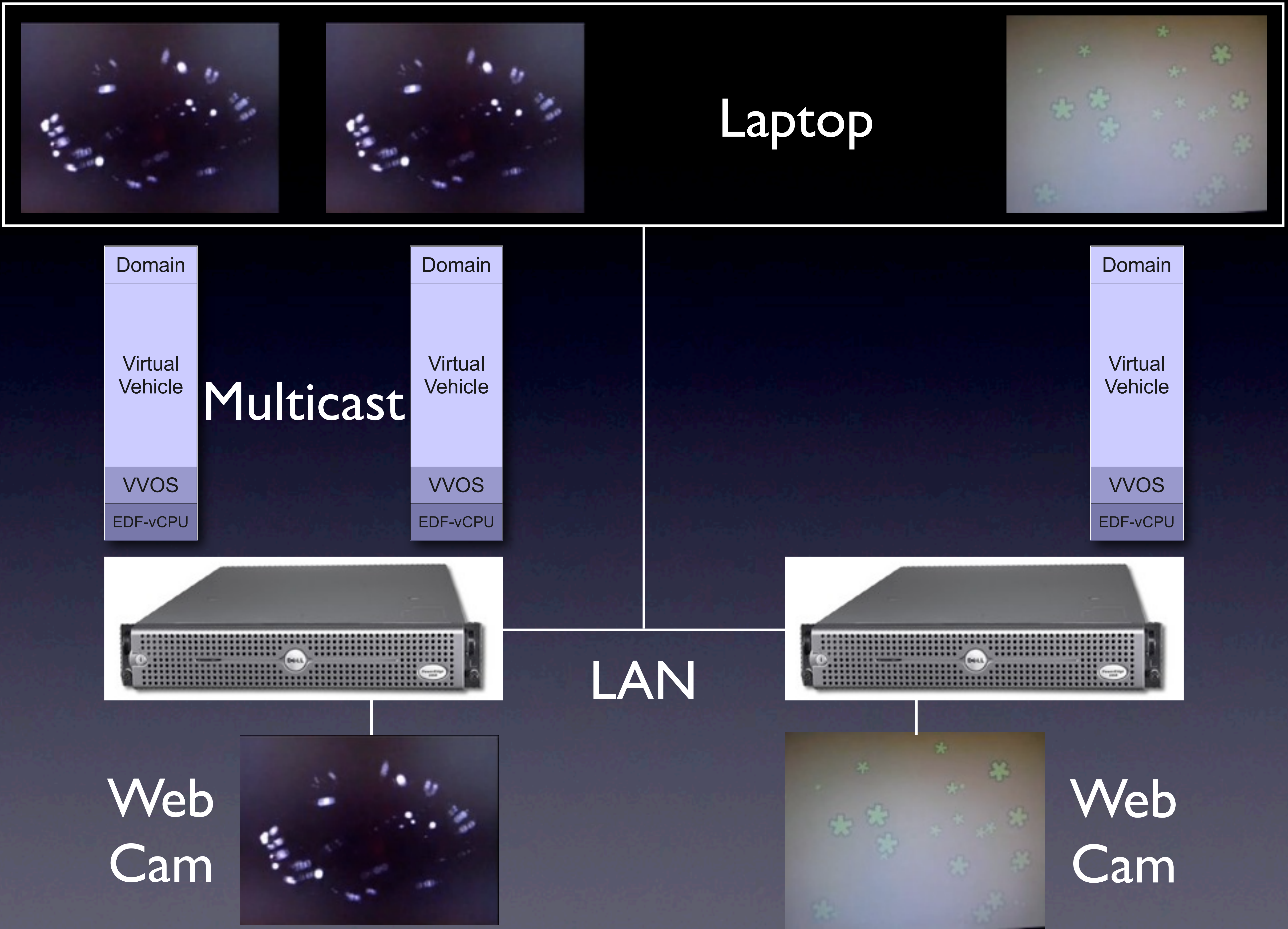
VVOS

EDF-vCPU

LAN

Web Cam

Web Cam



Laptop

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Multicast

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

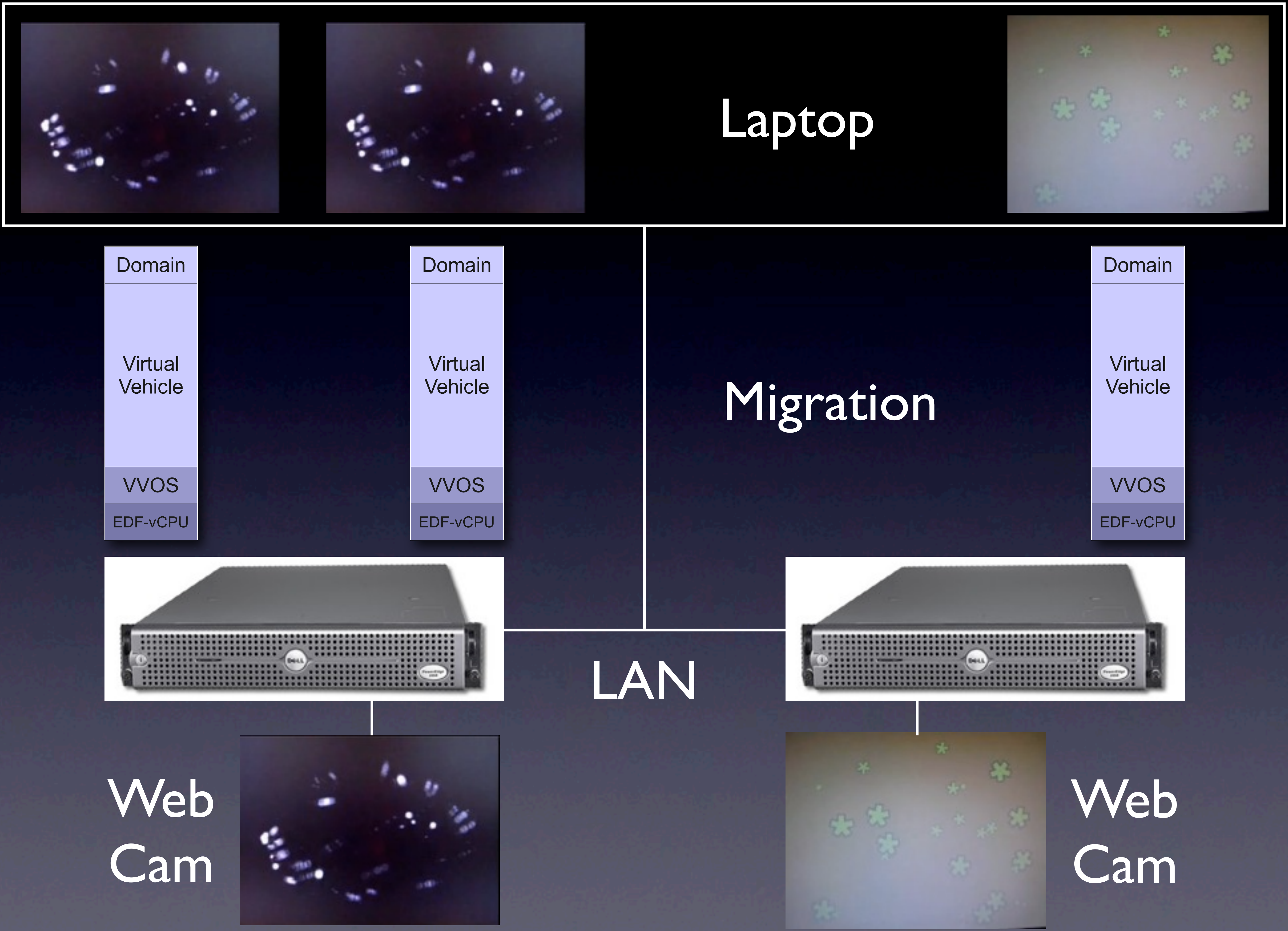
VVOS

EDF-vCPU

LAN

Web Cam

Web Cam



Laptop

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Migration

Domain

Virtual Vehicle

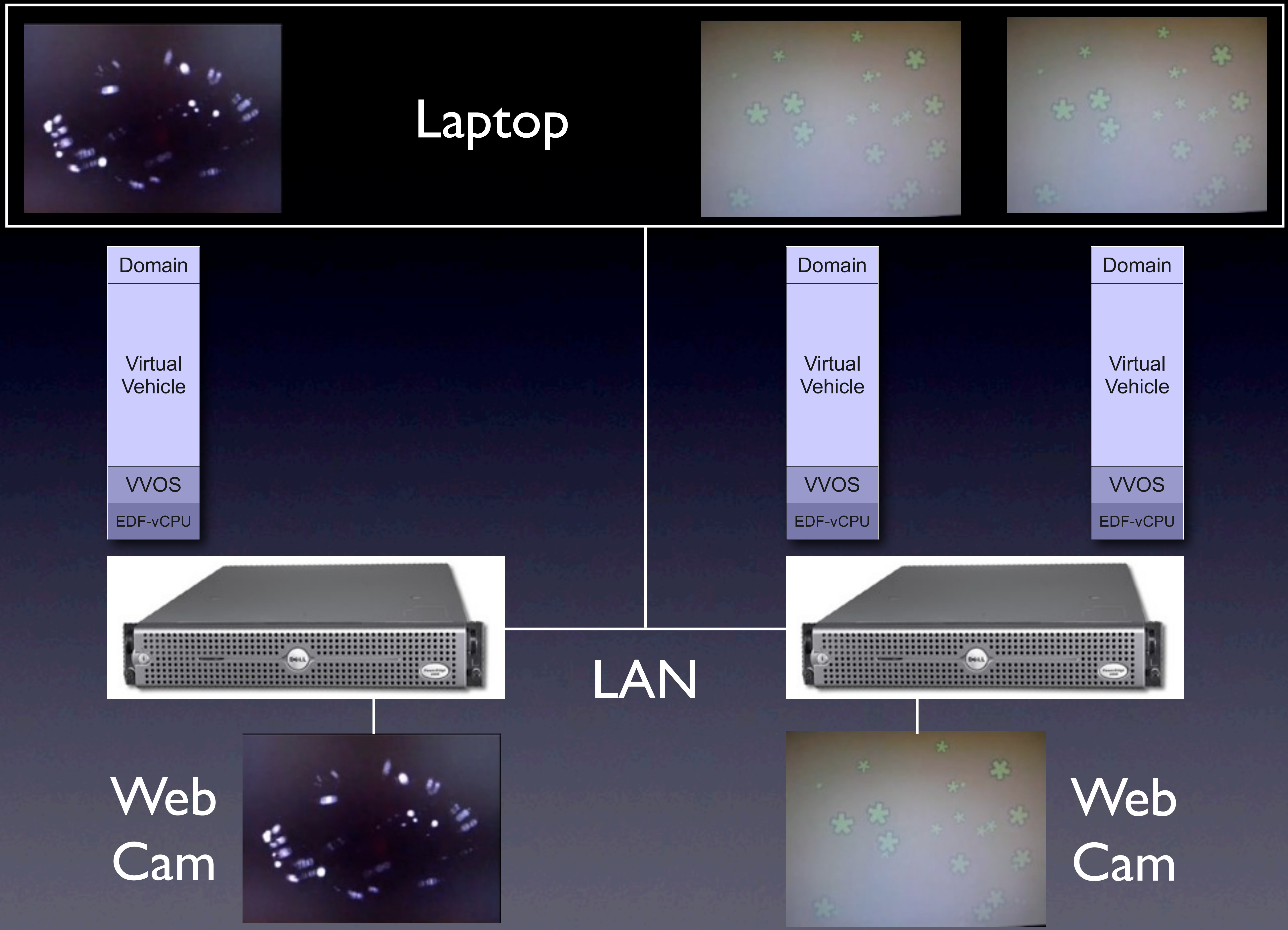
VVOS

EDF-vCPU

LAN

Web Cam

Web Cam



Laptop

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

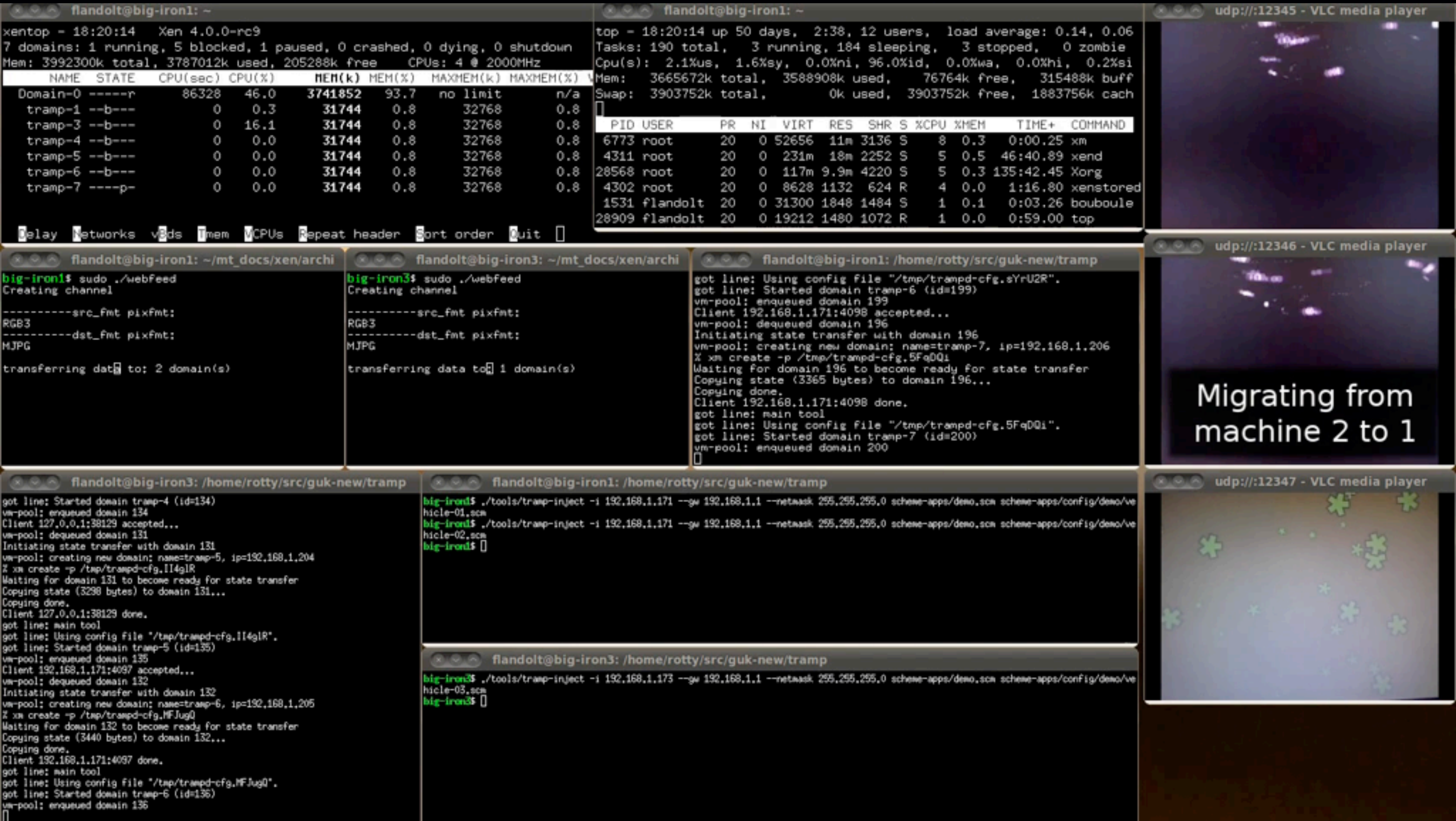
LAN

Web Cam

Web Cam

3 VVs on 2 Servers

3 VVs on 2 Servers


flandolt@big-iron1: ~
flandolt@big-iron1: ~
udp://:12345 - VLC media player
flandolt@big-iron1: ~/mt_docs/xen/archi
flandolt@big-iron3: ~/mt_docs/xen/archi
flandolt@big-iron1: /home/rotty/src/guk-new/tramp
udp://:12346 - VLC media player
flandolt@big-iron3: /home/rotty/src/guk-new/tramp
flandolt@big-iron1: /home/rotty/src/guk-new/tramp
udp://:12347 - VLC media player
flandolt@big-iron3: /home/rotty/src/guk-new/tramp

```
xentop - 18:20:14   Xen 4.0.0-rc9
7 domains: 1 running, 5 blocked, 1 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 3992300k total, 3787012k used, 205288k free    CPUs: 4 @ 2000MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%)
  Domain-0 -----r      86328   46.0    3741852   93.7   no limit       n/a
   tramp-1 --b---          0    0.3      31744    0.8      32768       0.8
   tramp-3 --b---          0   16.1      31744    0.8      32768       0.8
   tramp-4 --b---          0    0.0      31744    0.8      32768       0.8
   tramp-5 --b---          0    0.0      31744    0.8      32768       0.8
   tramp-6 --b---          0    0.0      31744    0.8      32768       0.8
   tramp-7 ----p-          0    0.0      31744    0.8      32768       0.8



Delay  Networks  vBds  Tmem  VCPUs  Repeat header  Sort order  Quit
```

```
top - 18:20:14 up 50 days,  2:38, 12 users,  load average: 0.14, 0.06
Tasks: 190 total,   3 running, 184 sleeping,   3 stopped,   0 zombie
Cpu(s):  2.1%us,  1.6%sy,  0.0%ni, 96.0%id,  0.0%wa,  0.0%hi,  0.2%si
Mem:   3665672k total,  3588908k used,    76764k free,   315488k buff
Swap:  3903752k total,        0k used,  3903752k free,  1883756k cach

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6773 root      20   0 52656  11m 3136 S    8  0.3   0:00.25 xm
 4311 root      20   0  231m  18m 2252 S    5  0.5  46:40.89 xend
28568 root      20   0  117m 9.9m 4220 S    5  0.3 135:42.45 Xorg
 4302 root      20   0  8628 1132  624 R    4  0.0   1:16.80 xenstored
 1531 flandolt  20   0 31300 1848 1484 S    1  0.1   0:03.26 bouboule
28909 flandolt  20   0 19212 1480 1072 R    1  0.0   0:59.00 top
```

```
big-iron1$ sudo ./webfeed
Creating channel

----------src_fmt pixfmt:
RGB3
----------dst_fmt pixfmt:
MJPG

transferring data to: 2 domain(s)
```

```
big-iron3$ sudo ./webfeed
Creating channel

----------src_fmt pixfmt:
RGB3
----------dst_fmt pixfmt:
MJPG

transferring data to: 1 domain(s)
```

```
got line: Using config file "/tmp/trampd-cfg.sYrU2R".
got line: Started domain tramp-6 (id=199)
vm-pool: enqueued domain 199
Client 192.168.1.171:4098 accepted...
vm-pool: dequeued domain 196
Initiating state transfer with domain 196
vm-pool: creating new domain: name=tramp-7, ip=192.168.1.206
% xm create -p /tmp/trampd-cfg.5FqDQi
Waiting for domain 196 to become ready for state transfer
Copying state (3365 bytes) to domain 196...
Copying done.
Client 192.168.1.171:4098 done.
got line: main tool
got line: Using config file "/tmp/trampd-cfg.5FqDQi".
got line: Started domain tramp-7 (id=200)
vm-pool: enqueued domain 200
```

Migrating from machine 2 to 1

```
got line: Started domain tramp-4 (id=134)
vm-pool: enqueued domain 134
Client 127.0.0.1:38129 accepted...
vm-pool: dequeued domain 131
Initiating state transfer with domain 131
vm-pool: creating new domain: name=tramp-5, ip=192.168.1.204
% xm create -p /tmp/trampd-cfg.II4glR
Waiting for domain 131 to become ready for state transfer
Copying state (3298 bytes) to domain 131...
Copying done.
Client 127.0.0.1:38129 done.
got line: main tool
got line: Using config file "/tmp/trampd-cfg.II4glR".
got line: Started domain tramp-5 (id=135)
vm-pool: enqueued domain 135
Client 192.168.1.171:4097 accepted...
vm-pool: dequeued domain 132
Initiating state transfer with domain 132
vm-pool: creating new domain: name=tramp-6, ip=192.168.1.205
% xm create -p /tmp/trampd-cfg.MFJugQ
Waiting for domain 132 to become ready for state transfer
Copying state (3440 bytes) to domain 132...
Copying done.
Client 192.168.1.171:4097 done.
got line: main tool
got line: Using config file "/tmp/trampd-cfg.MFJugQ".
got line: Started domain tramp-6 (id=136)
vm-pool: enqueued domain 136
```

```
big-iron1$ ./tools/tramp-inject -i 192.168.1.171 --gw 192.168.1.1 --netmask 255.255.255.0 scheme-apps/demo.scm scheme-apps/config/demo/ve
hicle-01.scm
big-iron1$ ./tools/tramp-inject -i 192.168.1.171 --gw 192.168.1.1 --netmask 255.255.255.0 scheme-apps/demo.scm scheme-apps/config/demo/ve
hicle-02.scm
big-iron1$
```

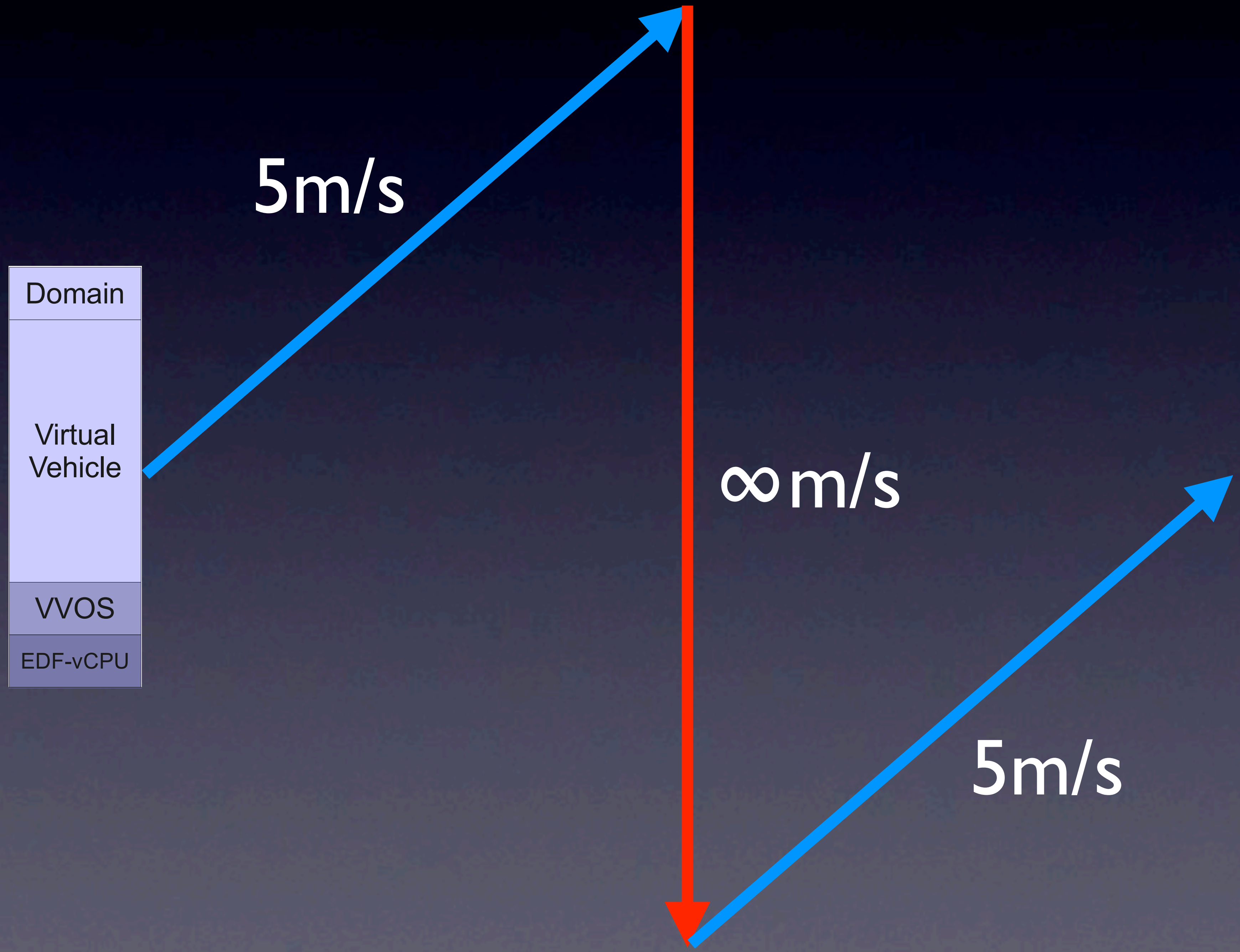
```
big-iron3$ ./tools/tramp-inject -i 192.168.1.173 --gw 192.168.1.1 --netmask 255.255.255.0 scheme-apps/demo.scm scheme-apps/config/demo/ve
hicle-03.scm
big-iron3$
```

Goals and Challenges

- Multi-provider (10s):
  - heterogeneous operations
- Multi-vehicle (100s):
  - heterogeneous systems
- Multi-task (1000s):
  - heterogeneous missions

- Programming Language
  - Berkeley, Salzburg
- Collaborative Control
  - Berkeley
- Virtualization Infrastructure
  - Salzburg

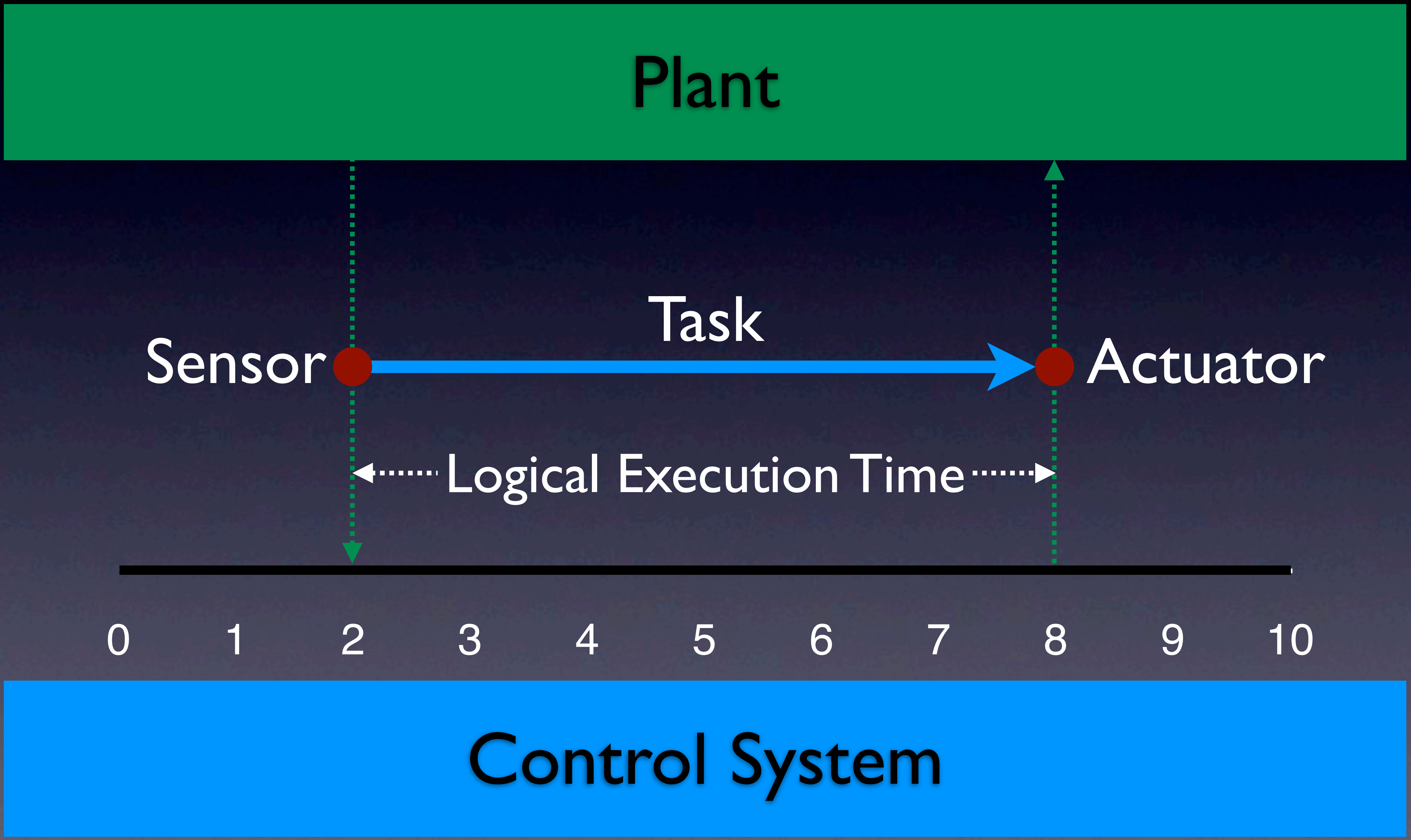
“Logical Execution Space”



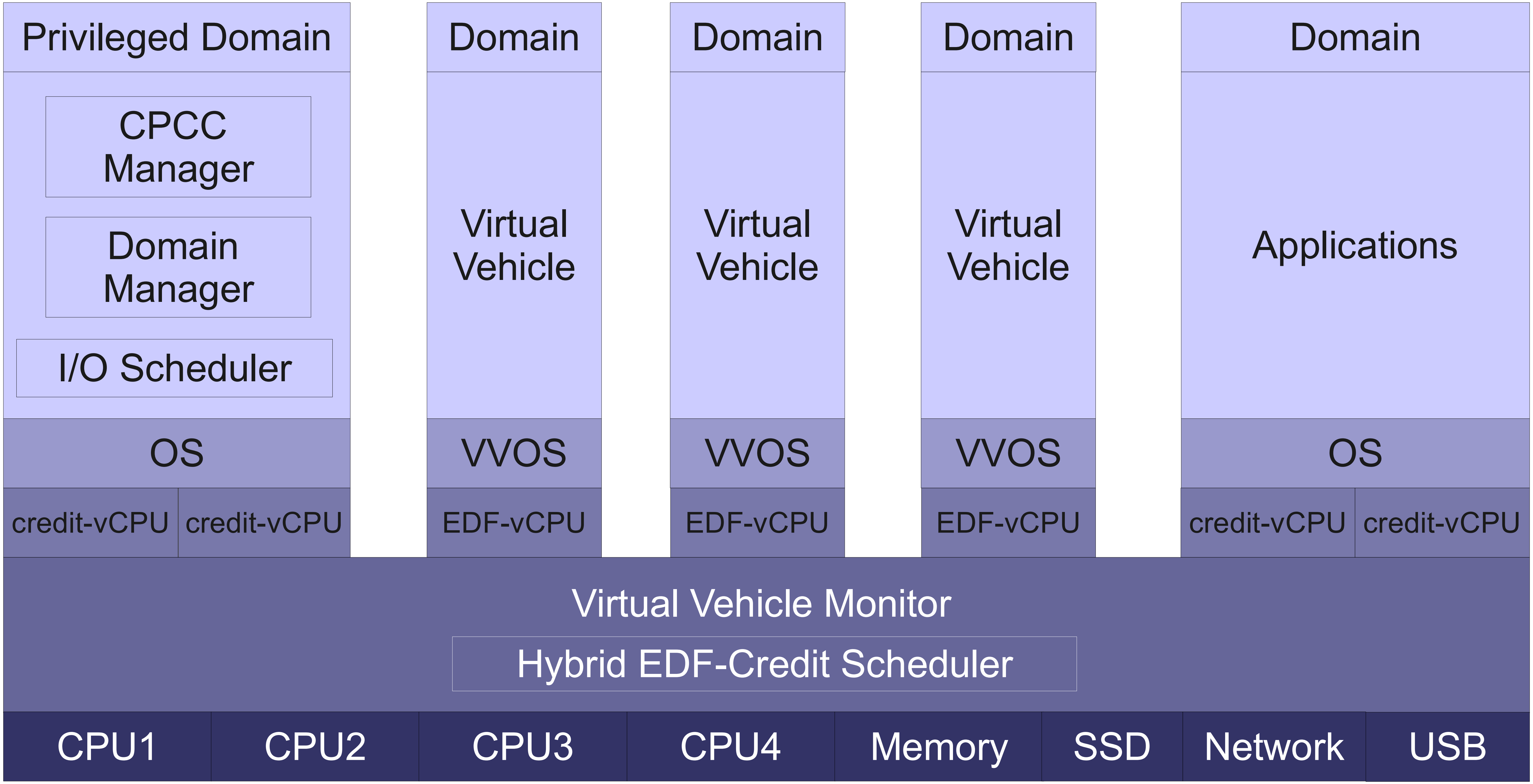
© C. Kirsch 2011

Logical Execution Time

[EMSOFT01, PLDI02, Proc. of the IEEE03, TOPLAS07]



Virtualization Infrastructure



Privileged Domain

CPCC Manager

Domain Manager

I/O Scheduler

OS

credit-vCPU credit-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Virtual Vehicle

VVOS

EDF-vCPU

Domain

Applications

OS

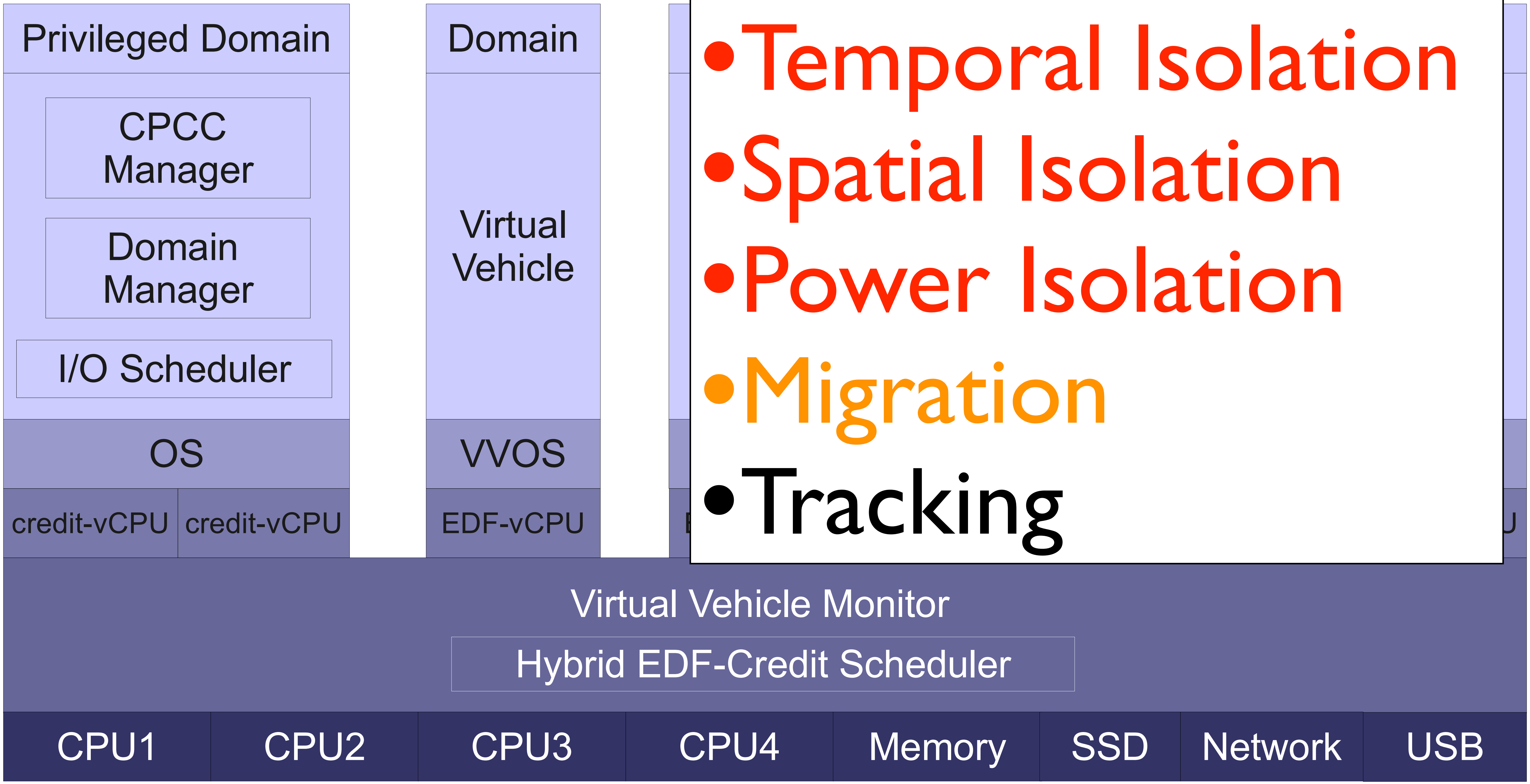
credit-vCPU credit-vCPU

Virtual Vehicle Monitor

Hybrid EDF-Credit Scheduler

CPU1 CPU2 CPU3 CPU4 Memory SSD Network USB

Virtualization Infrastructure



- Temporal Isolation
- Spatial Isolation
- Power Isolation
- Migration
- Tracking

There is a

fundamental trade-off

between

quality and cost

of

time, space, power

isolation

Time

[SIES09,RTAS10]

- quality: response time jitter
- cost: scheduling overhead

Time

[SIES09,RTAS10]

- quality: response time jitter
- cost: scheduling overhead

Space

[USENIX ATC08,ISMM11]

- quality: fragmentation jitter
- cost: management overhead

Time

[SIES09,RTAS10]

- quality: response time jitter
- cost: scheduling overhead

Space

[USENIX ATC08,ISMM11]

- quality: fragmentation jitter
- cost: management overhead

Power

[EMSOFT10]

- quality: power consumption jitter
- cost: total power consumption

1. Memory Management:
Short-term Memory

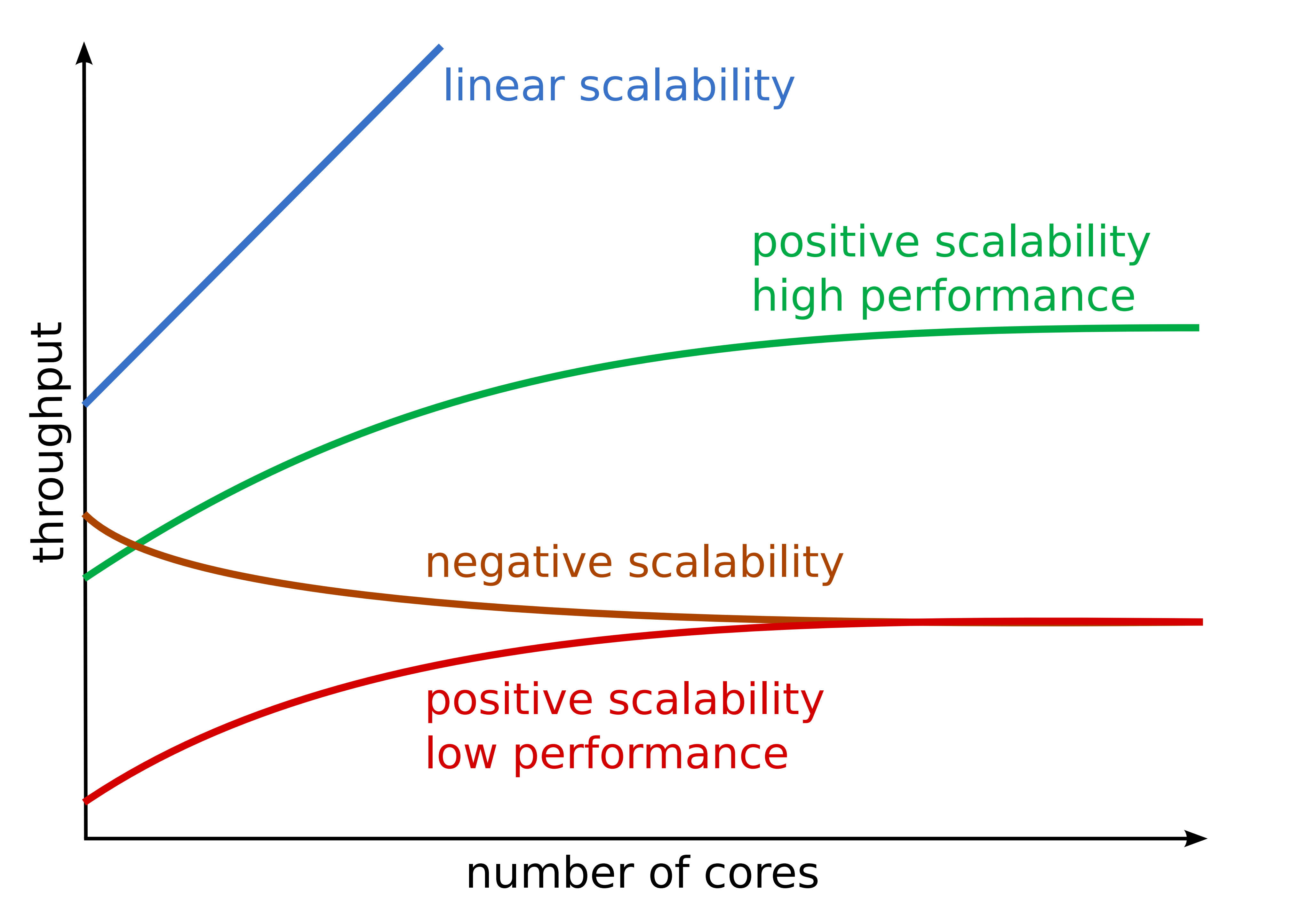
2. Concurrency Management:
Non-deterministic Data Structures

Performance, Scalability, and Semantics of Concurrent FIFO Queues

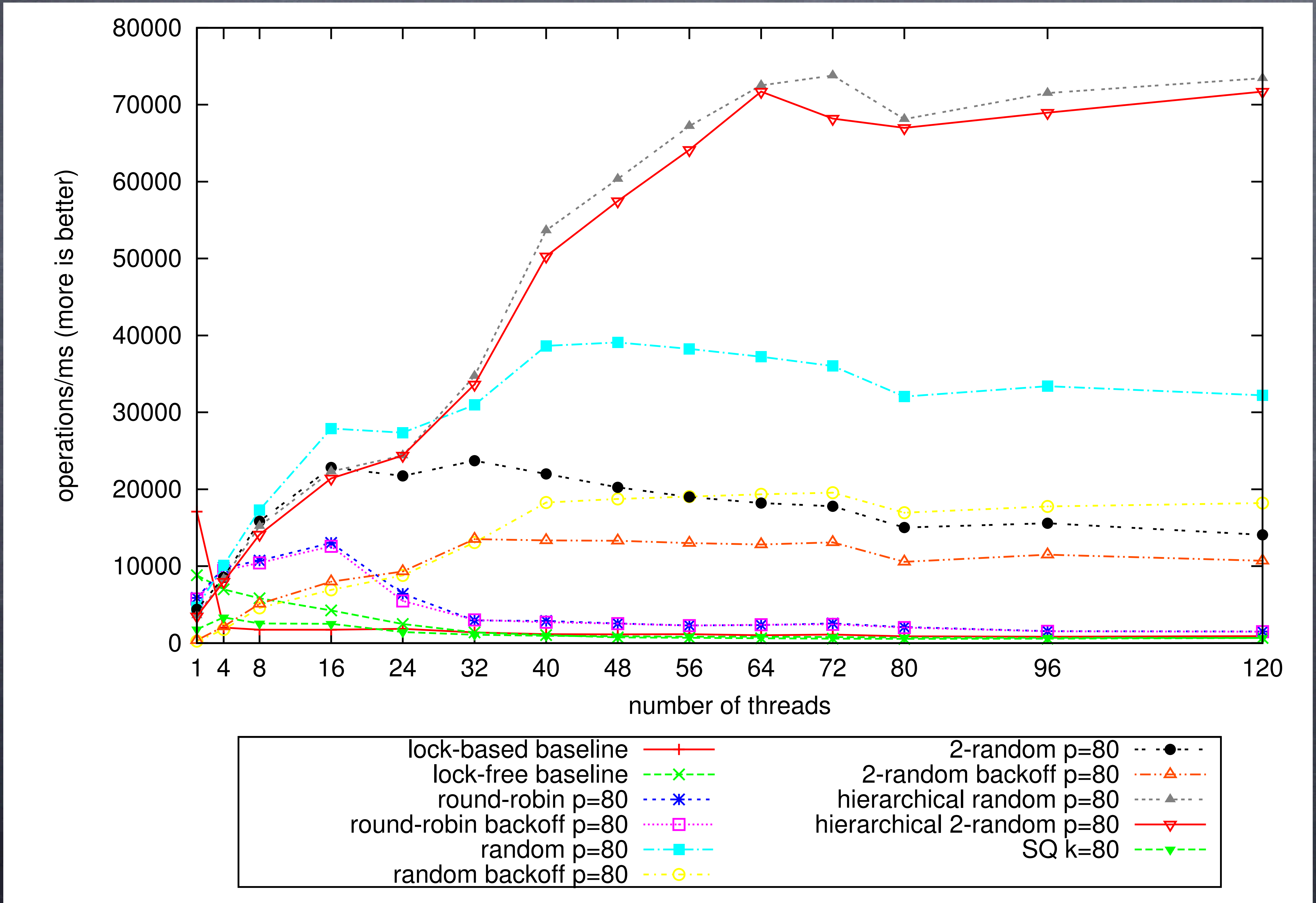
Christoph Kirsch, Hannes Payer,
Harald Röck, Ana Sokolova

Universität Salzburg

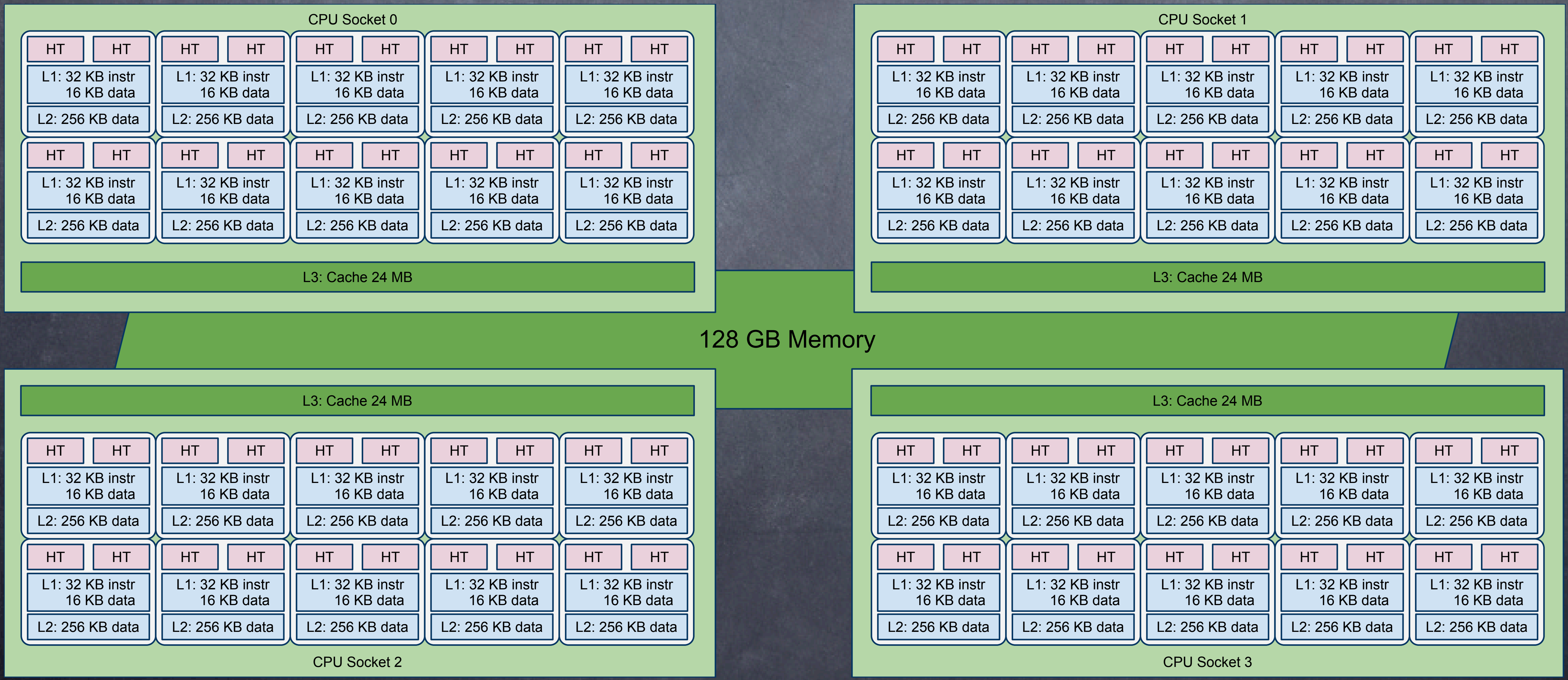
Performance & Scalability



High Contention



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



CPU Socket 0

HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data

L3: Cache 24 MB

CPU Socket 1

HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data

L3: Cache 24 MB

128 GB Memory

L3: Cache 24 MB

HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data

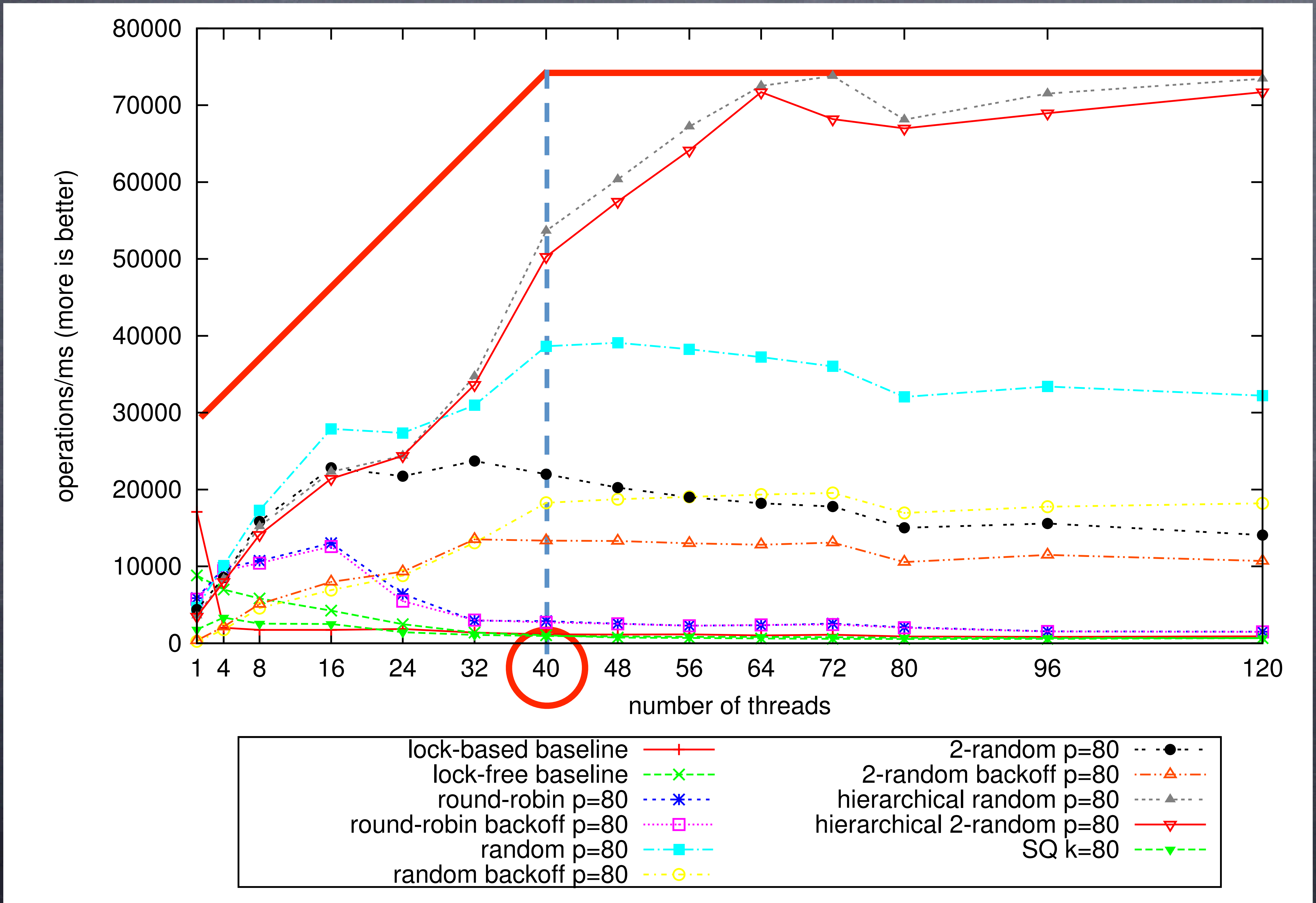
CPU Socket 2

L3: Cache 24 MB

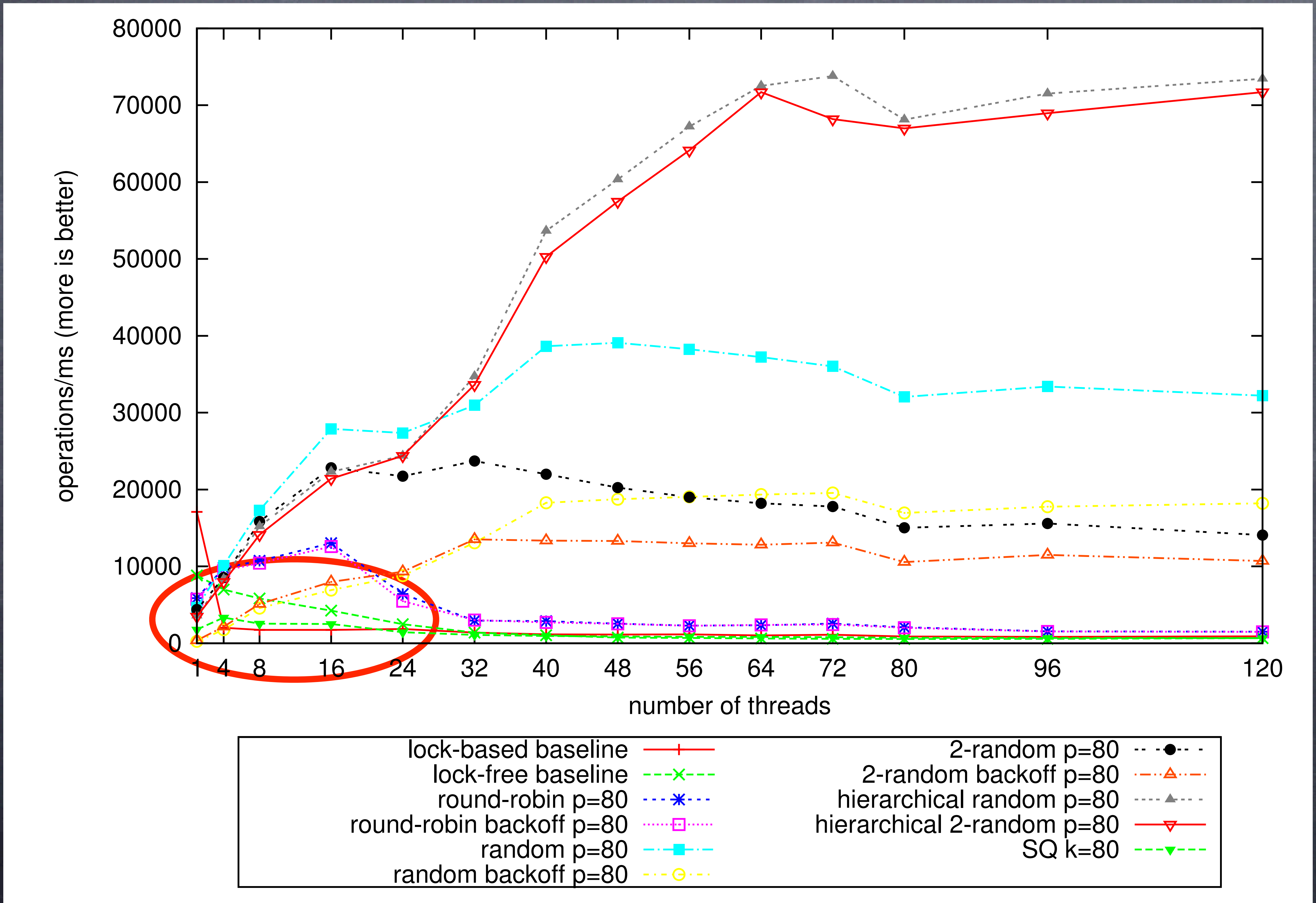
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data
HT HT L1: 32 KB instr 16 KB data L2: 256 KB data

CPU Socket 3

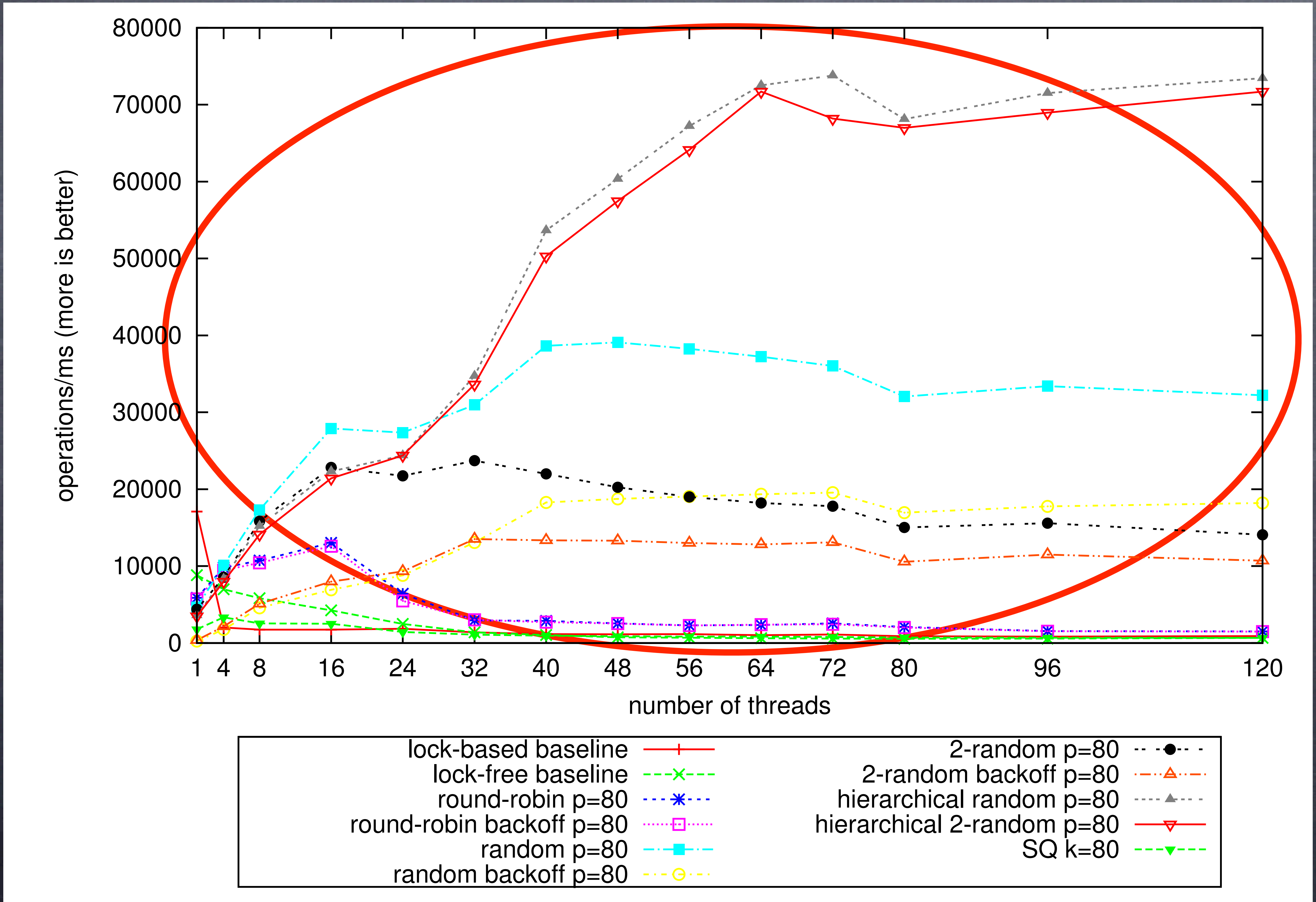
Ideal 40-Core Performance



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]

Regular FIFO

Michael-Scott (MS)
[MS96]

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)

Flat Combining (FC)
[IST10]

Regular FIFO

Michael-Scott (MS)
[MS96]

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)

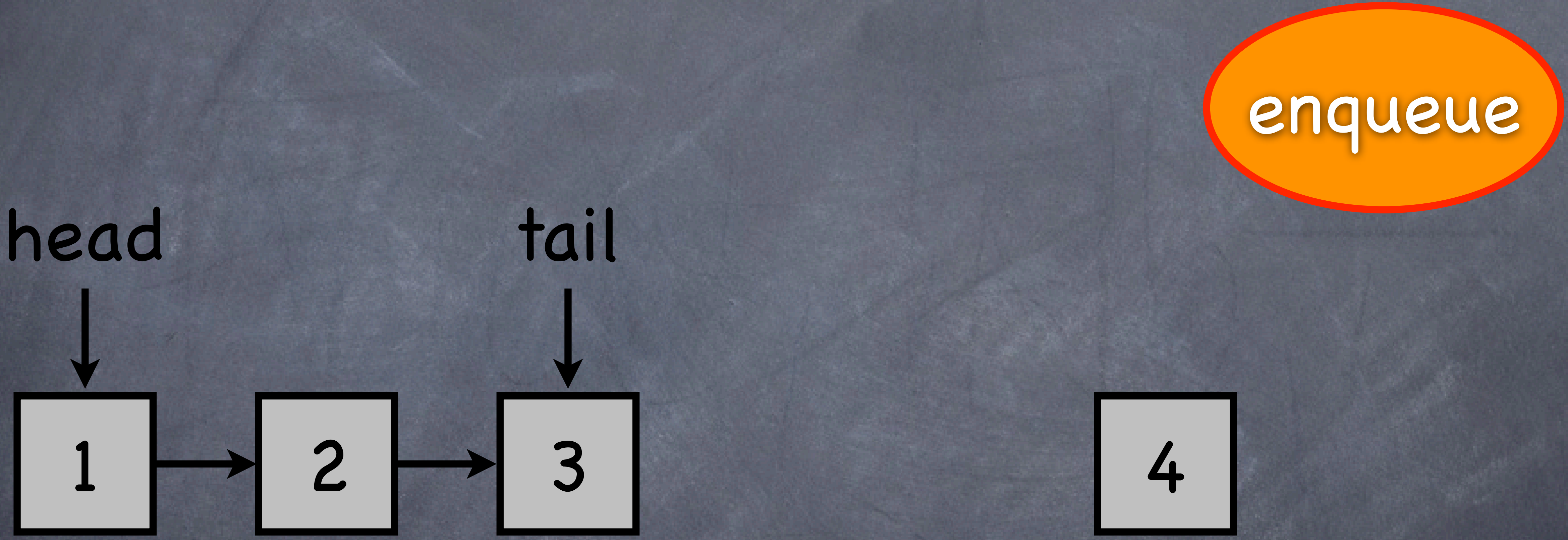
k-FIFO Queues

- 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

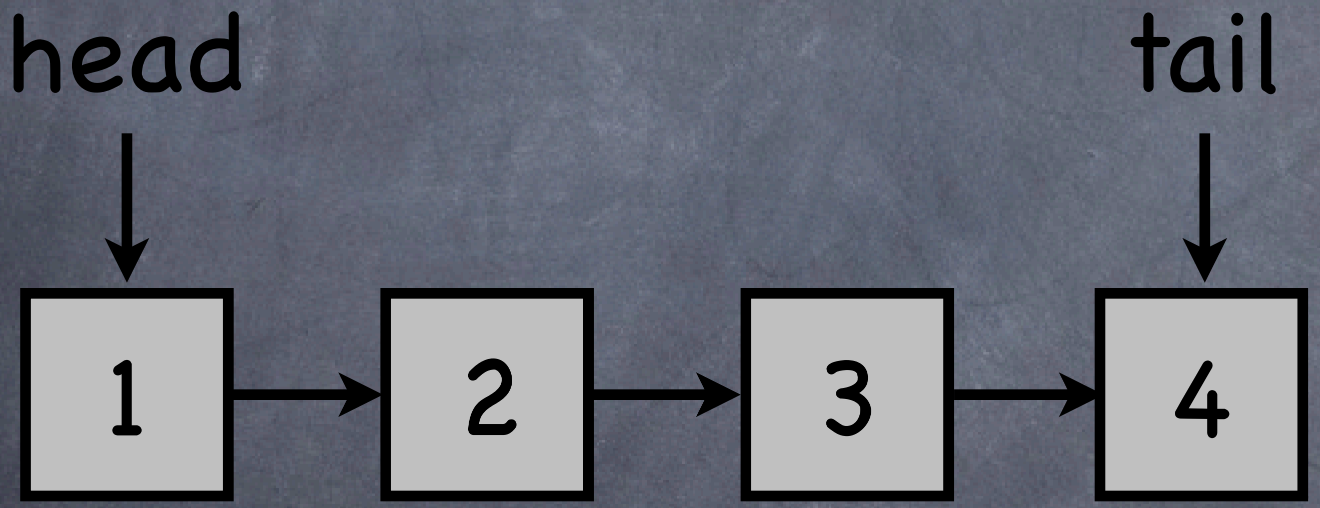
k-FIFO Queues

- 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

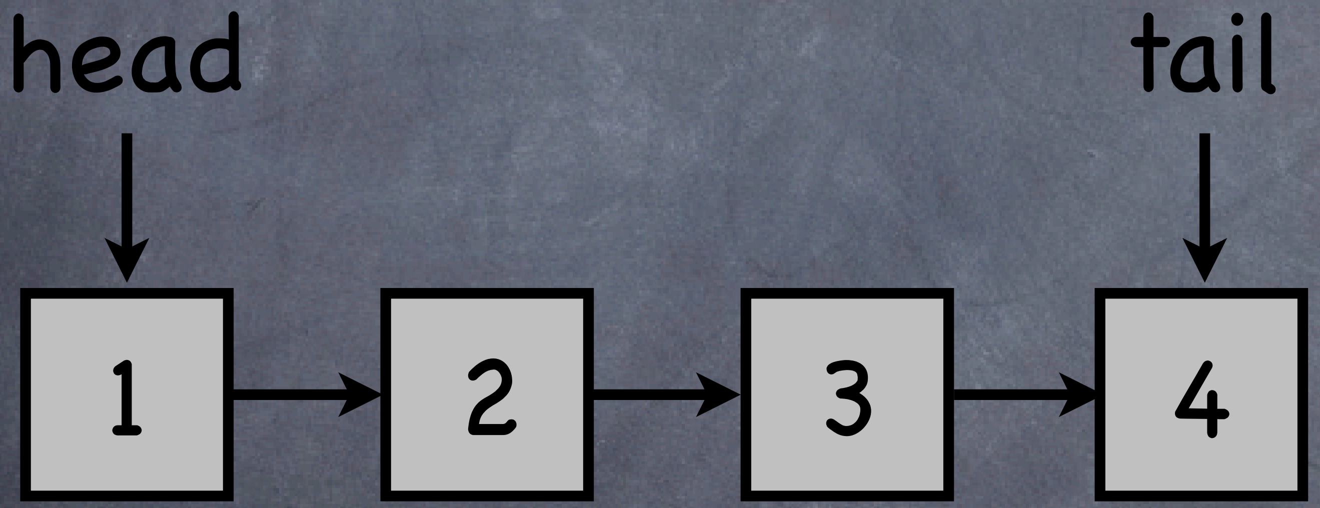


Example: k=2



enqueue

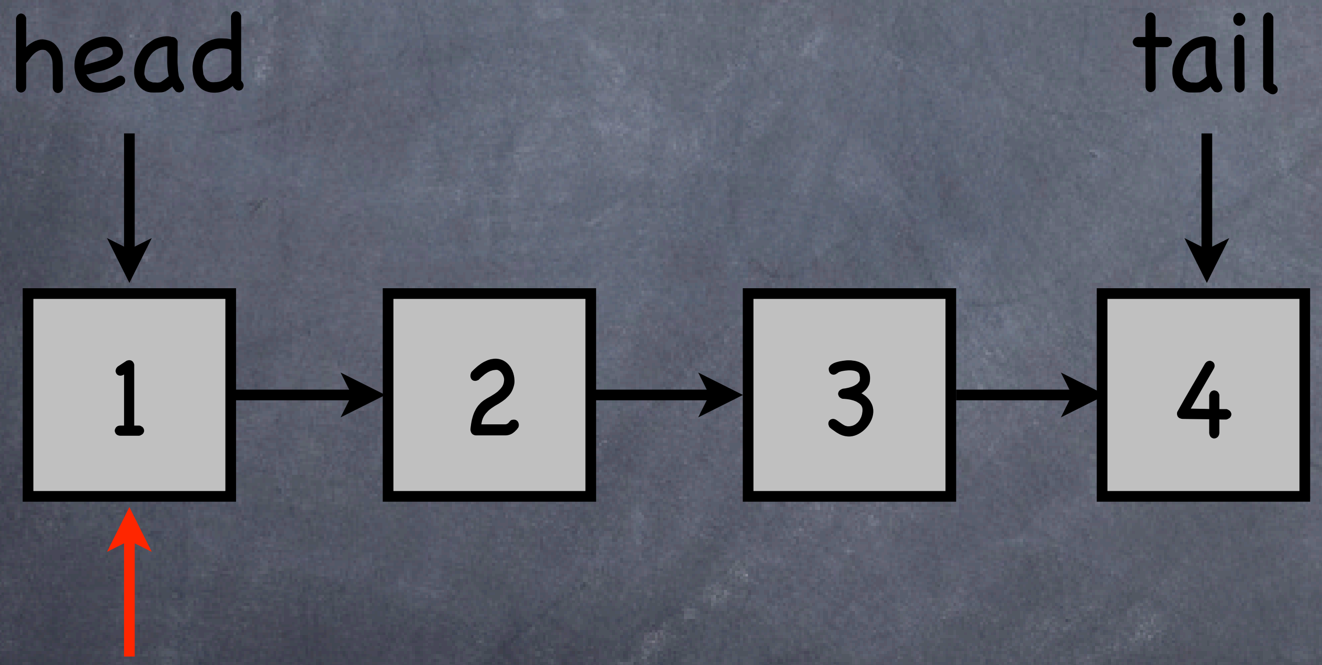
Example: k=2



dequeue

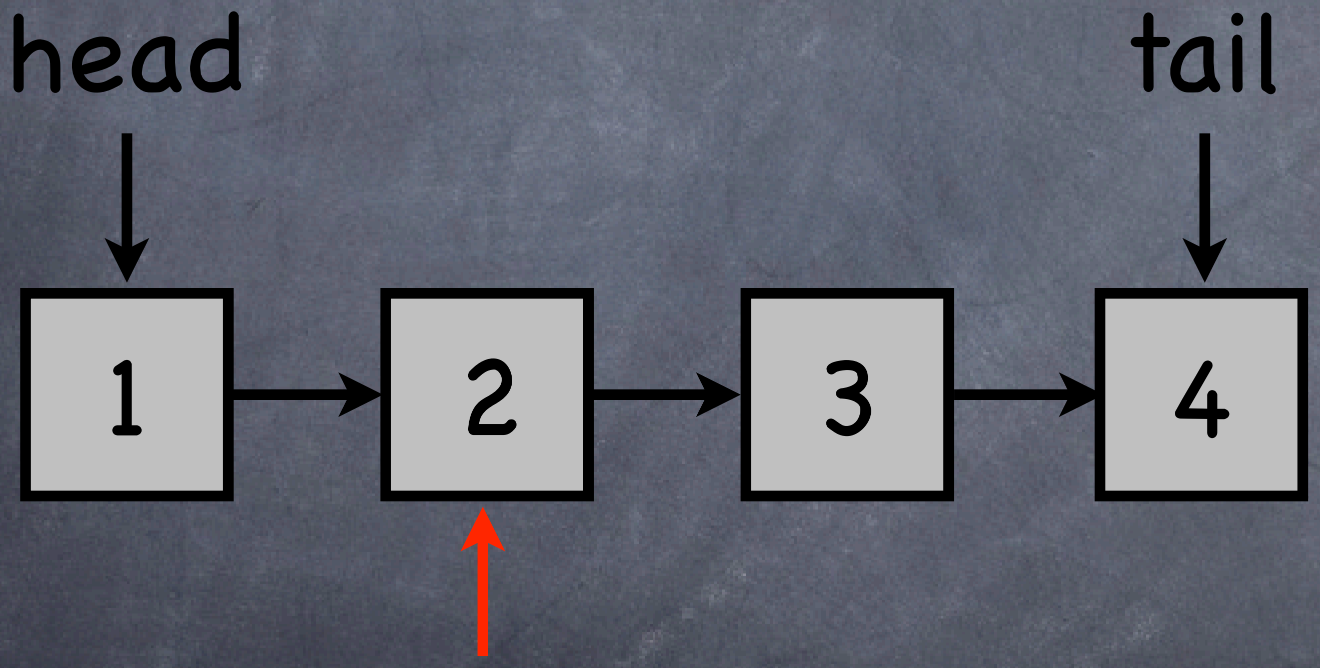
Example: k=2

dequeue

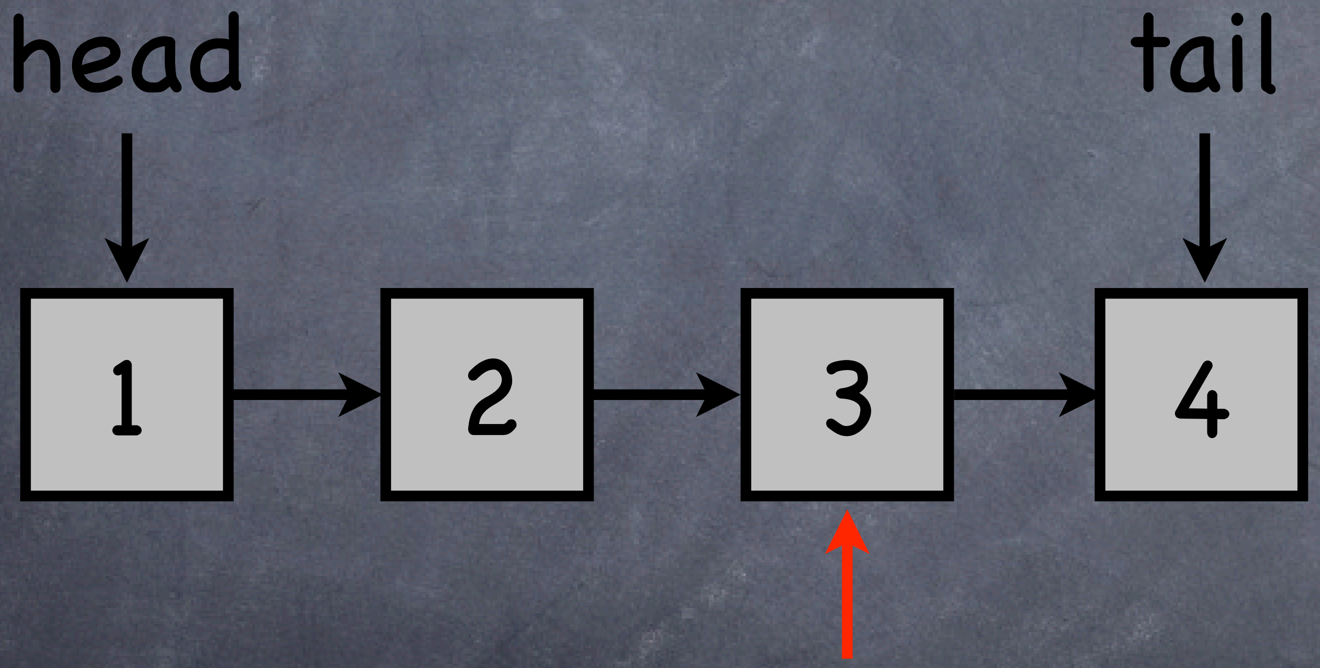


Example: k=2

dequeue

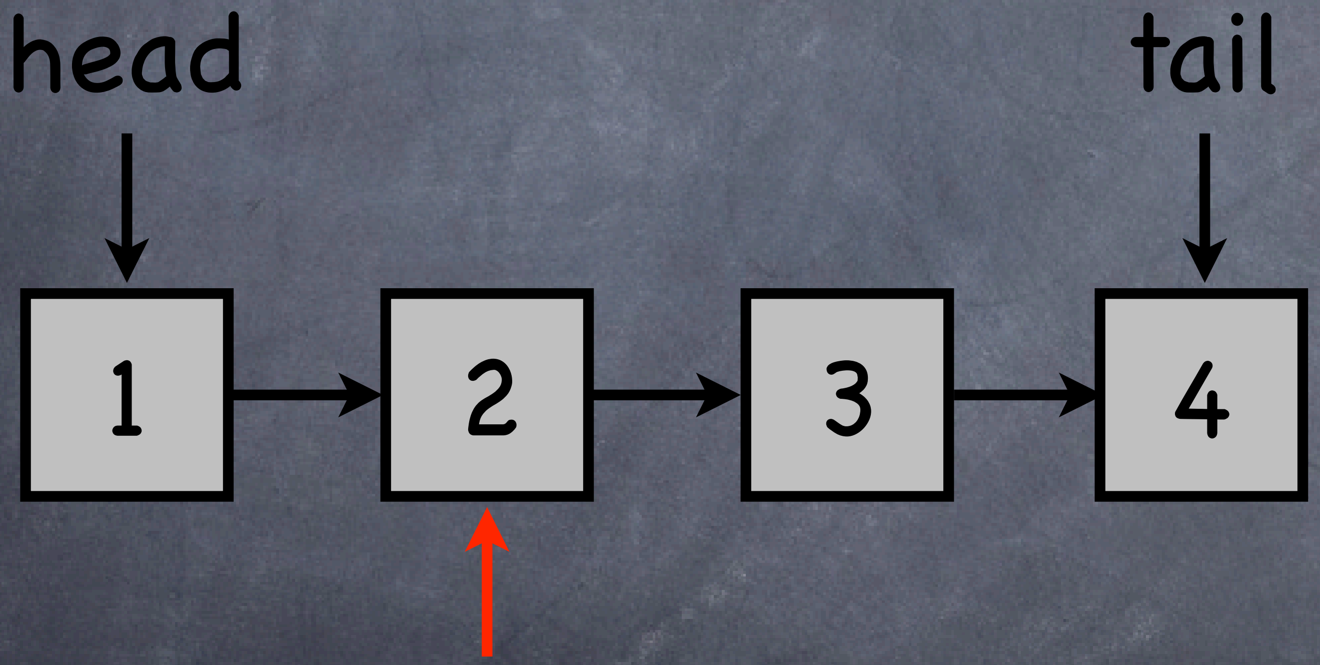


Example: k=2



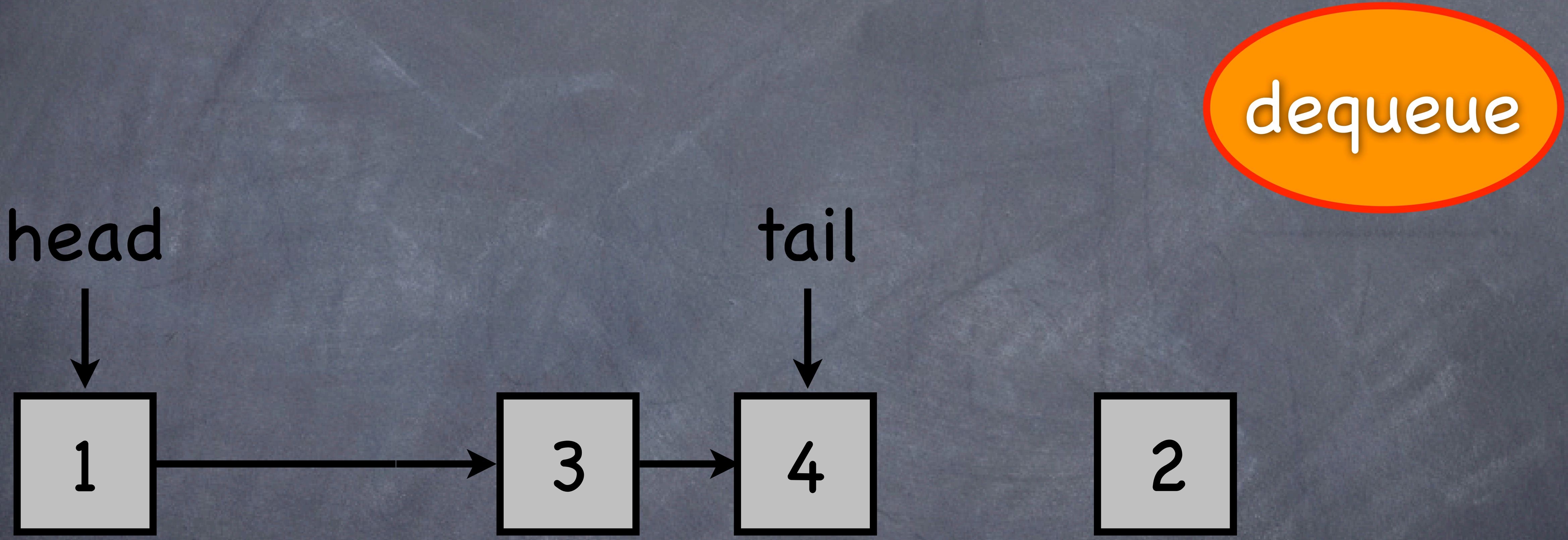
dequeue

Example: k=2

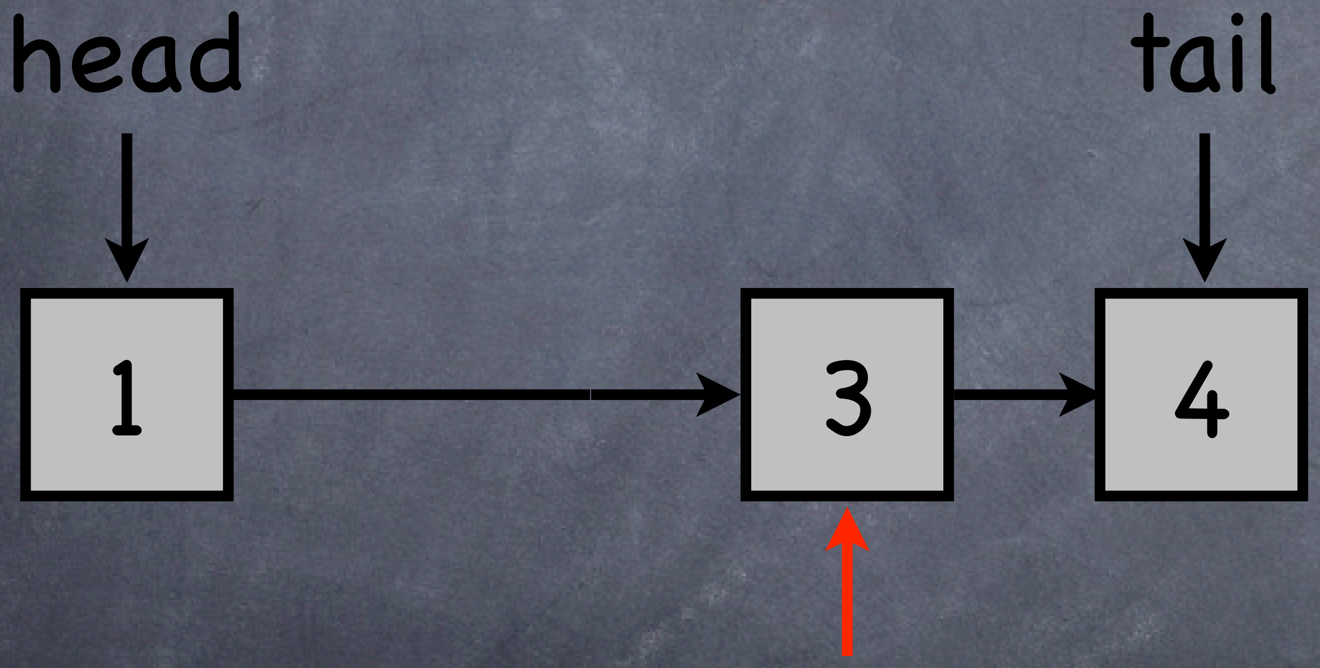


dequeue

Example: k=2

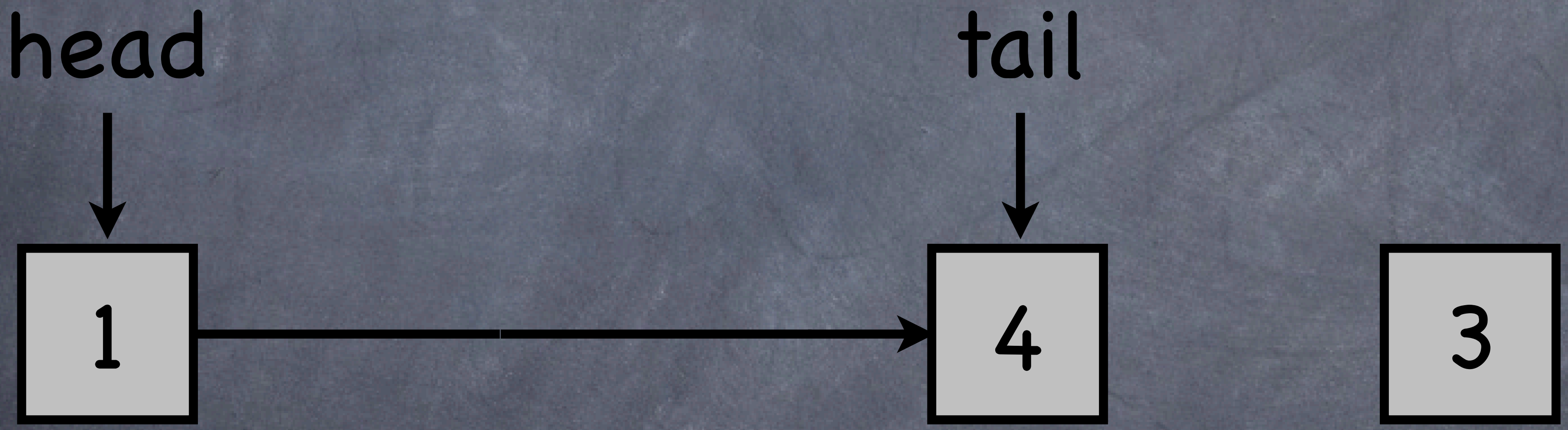


Example: k=2



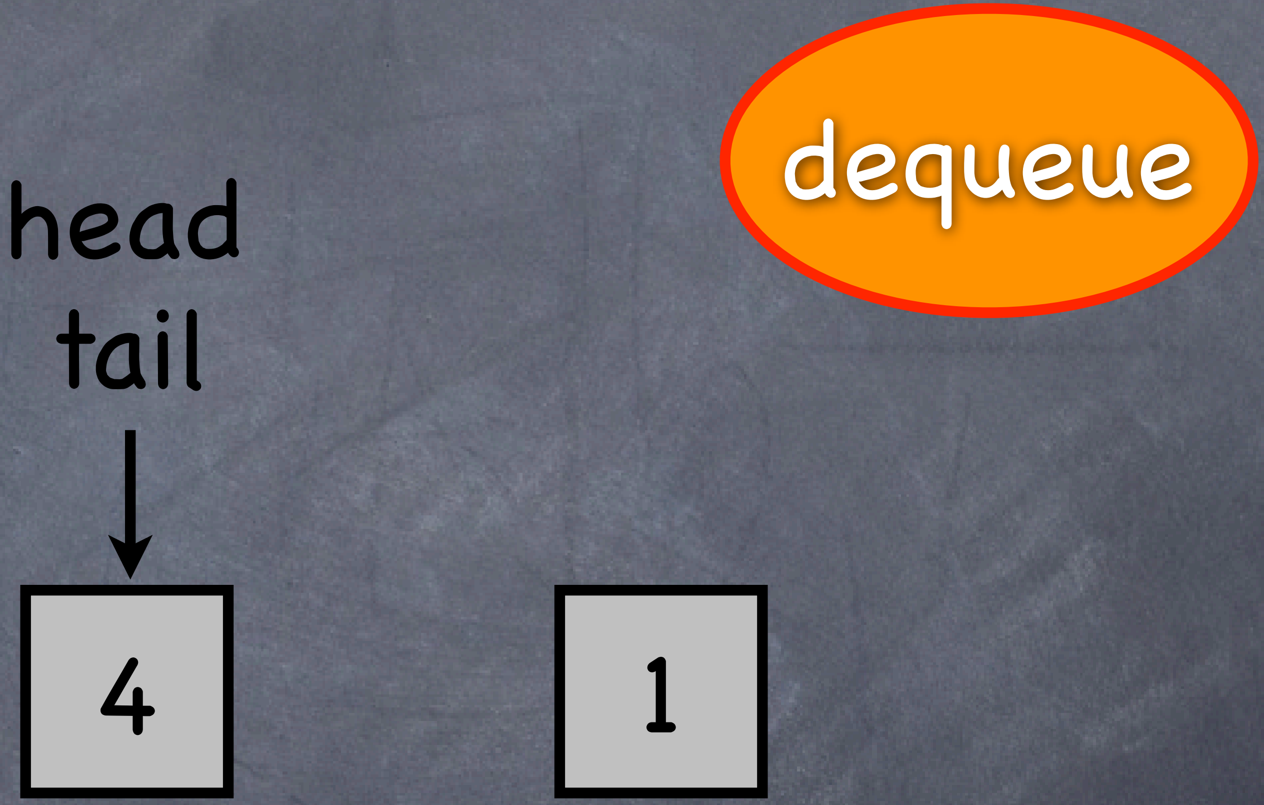
dequeue

Example: k=2



dequeue

Example: k=2



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.

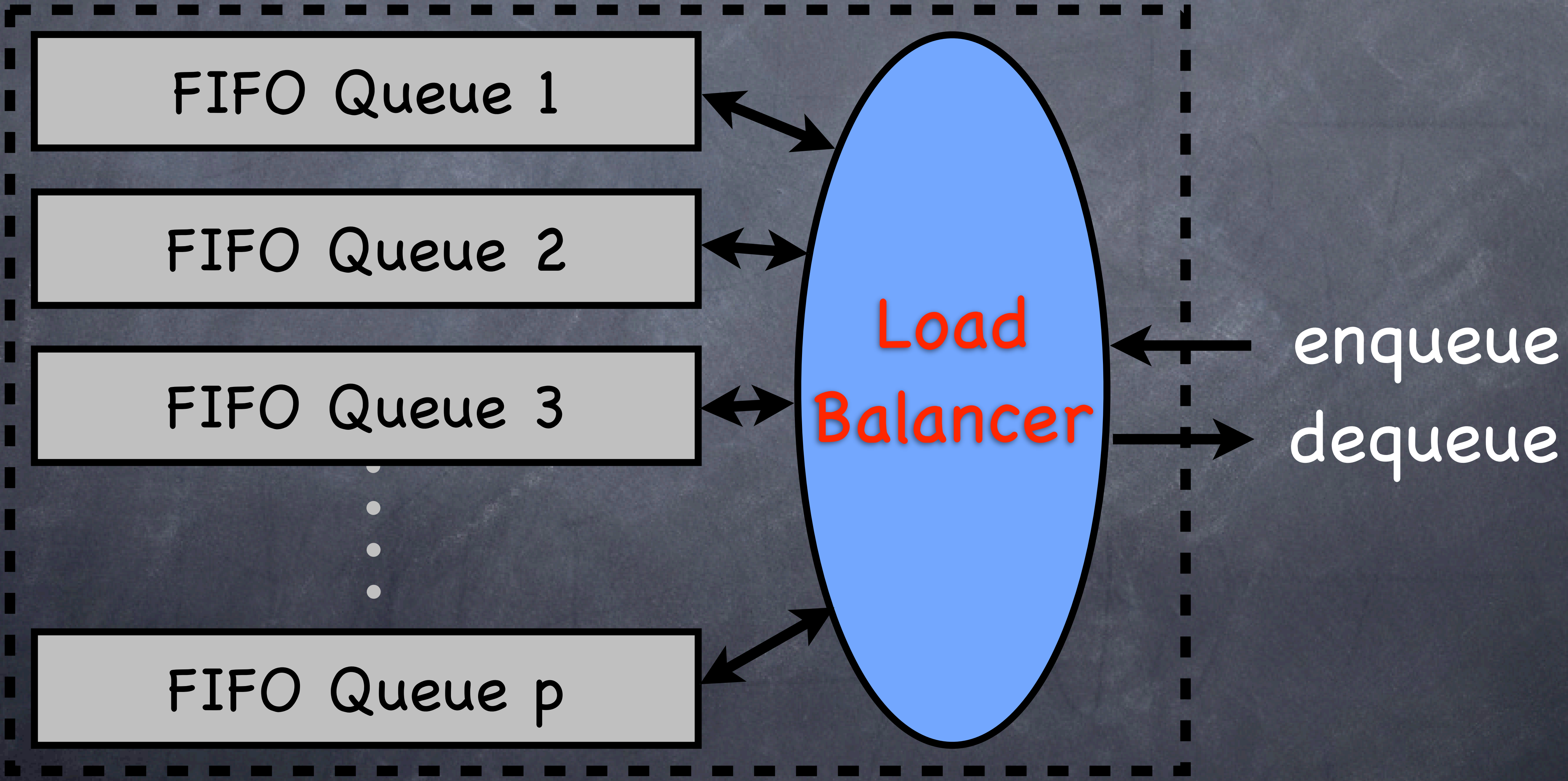
WCSD of existing k-FIFO Queue Implementations

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

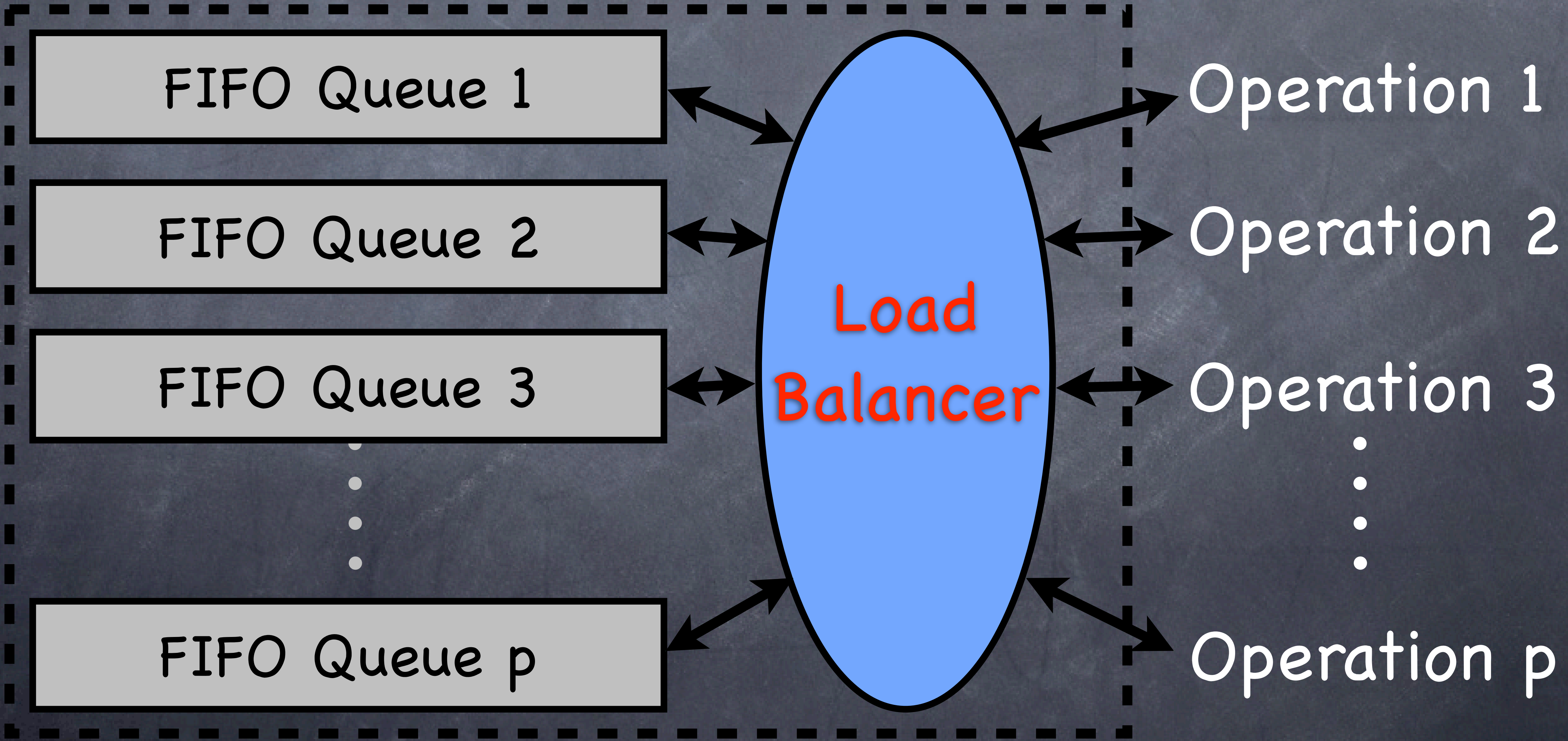
workload-independent constant bounds

[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.

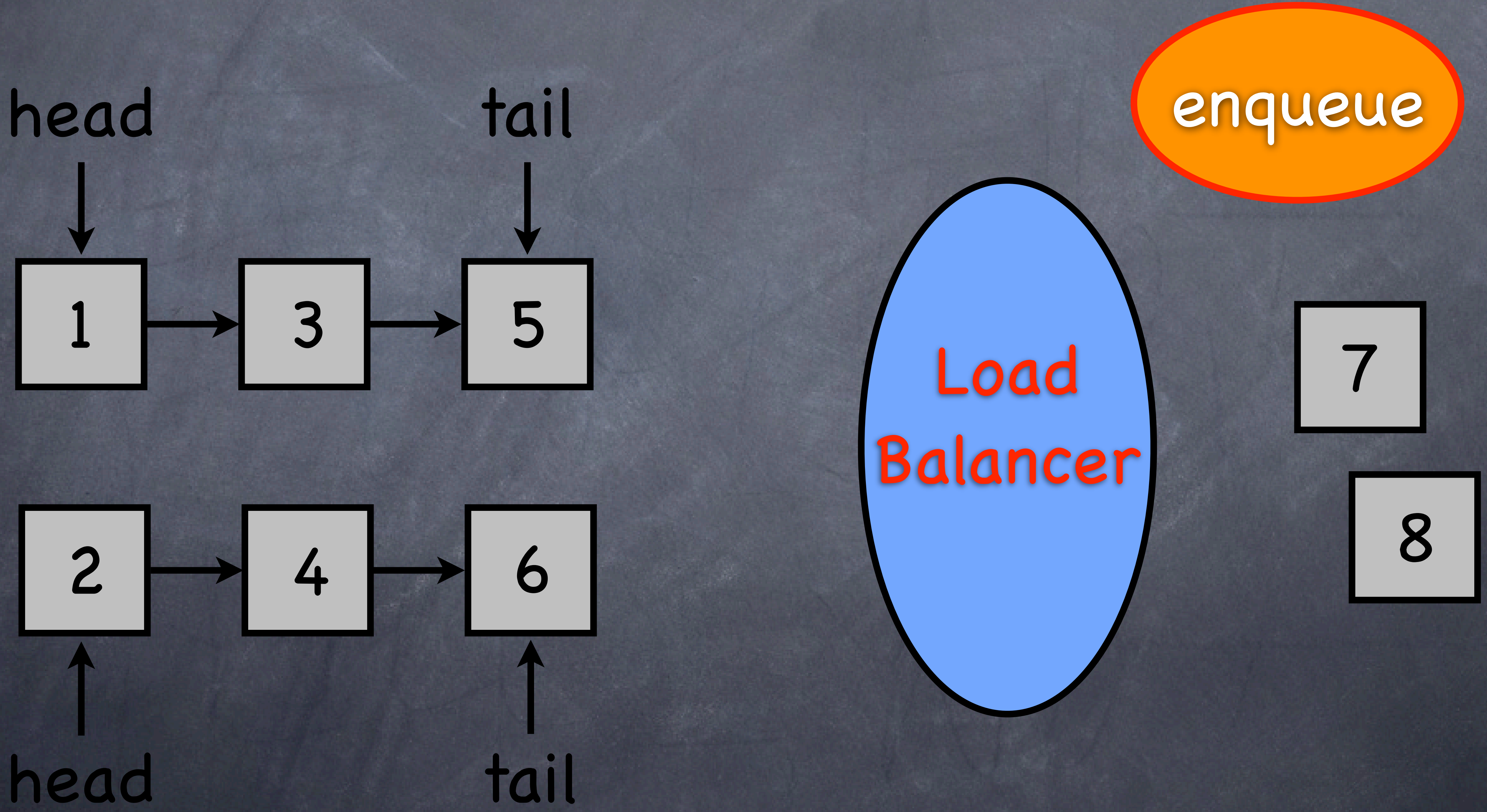
Scal Queue:
p FIFO Queues



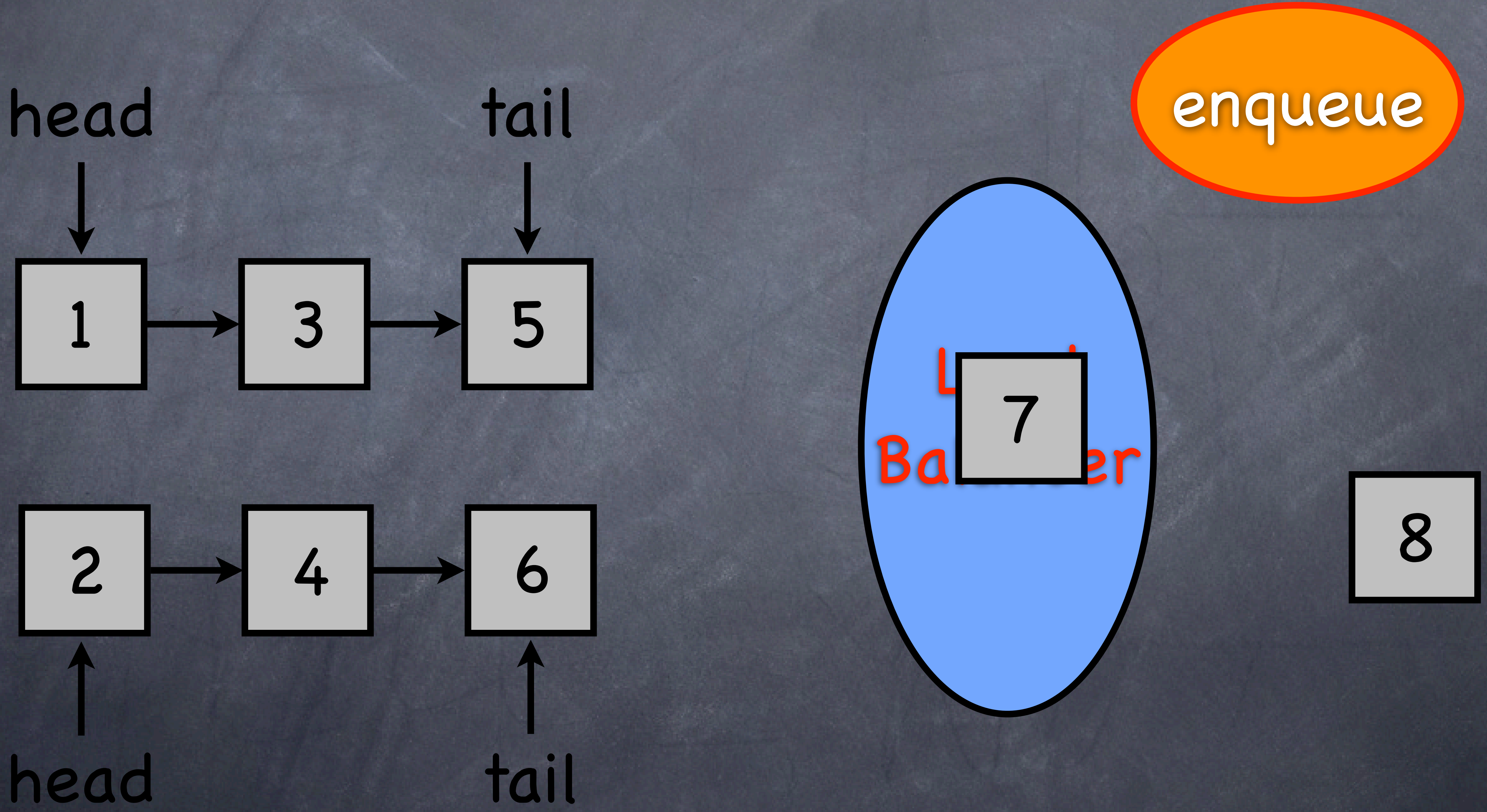
Scal Queue: Up to p Parallel Operations



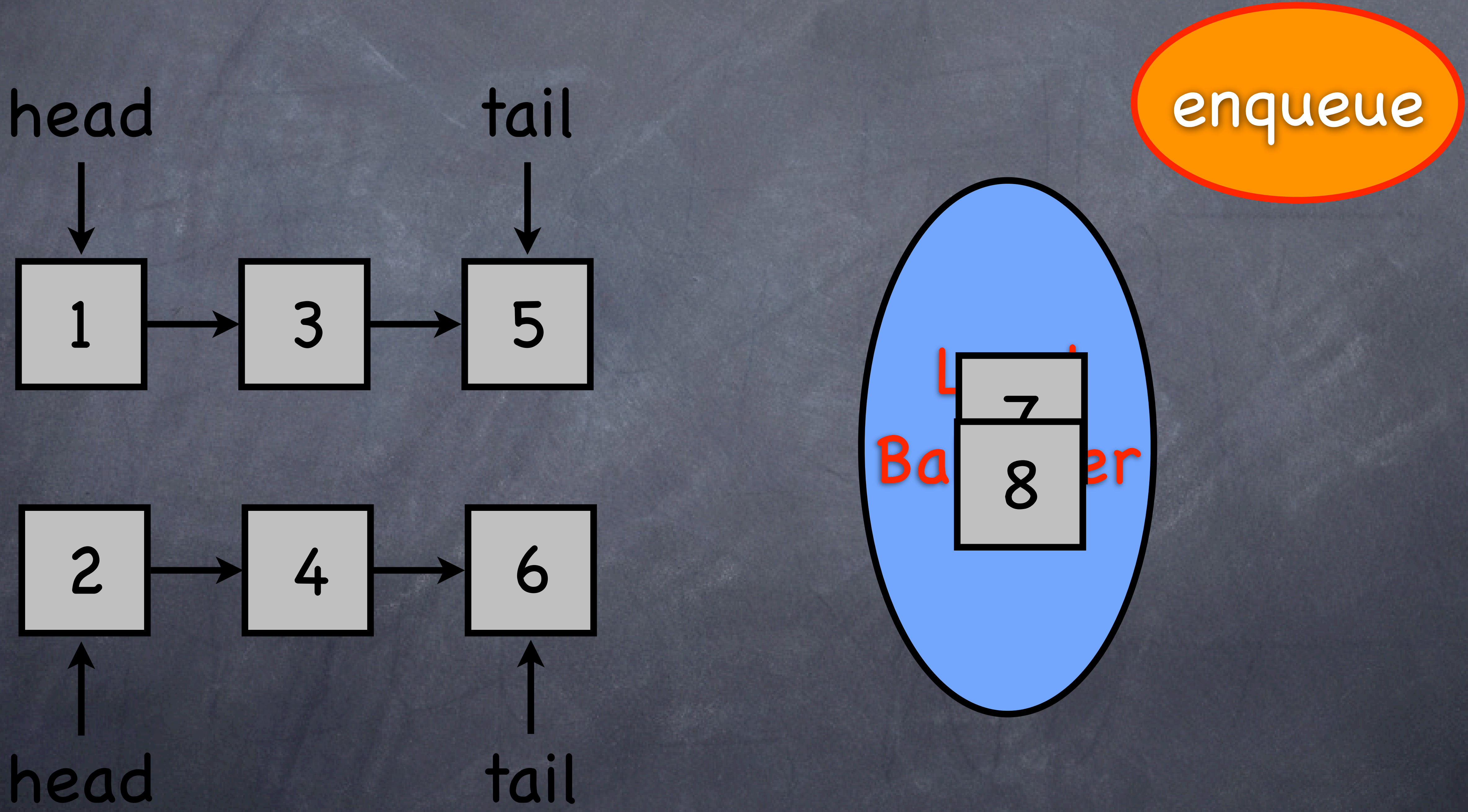
Scal Queue: Load Balancing



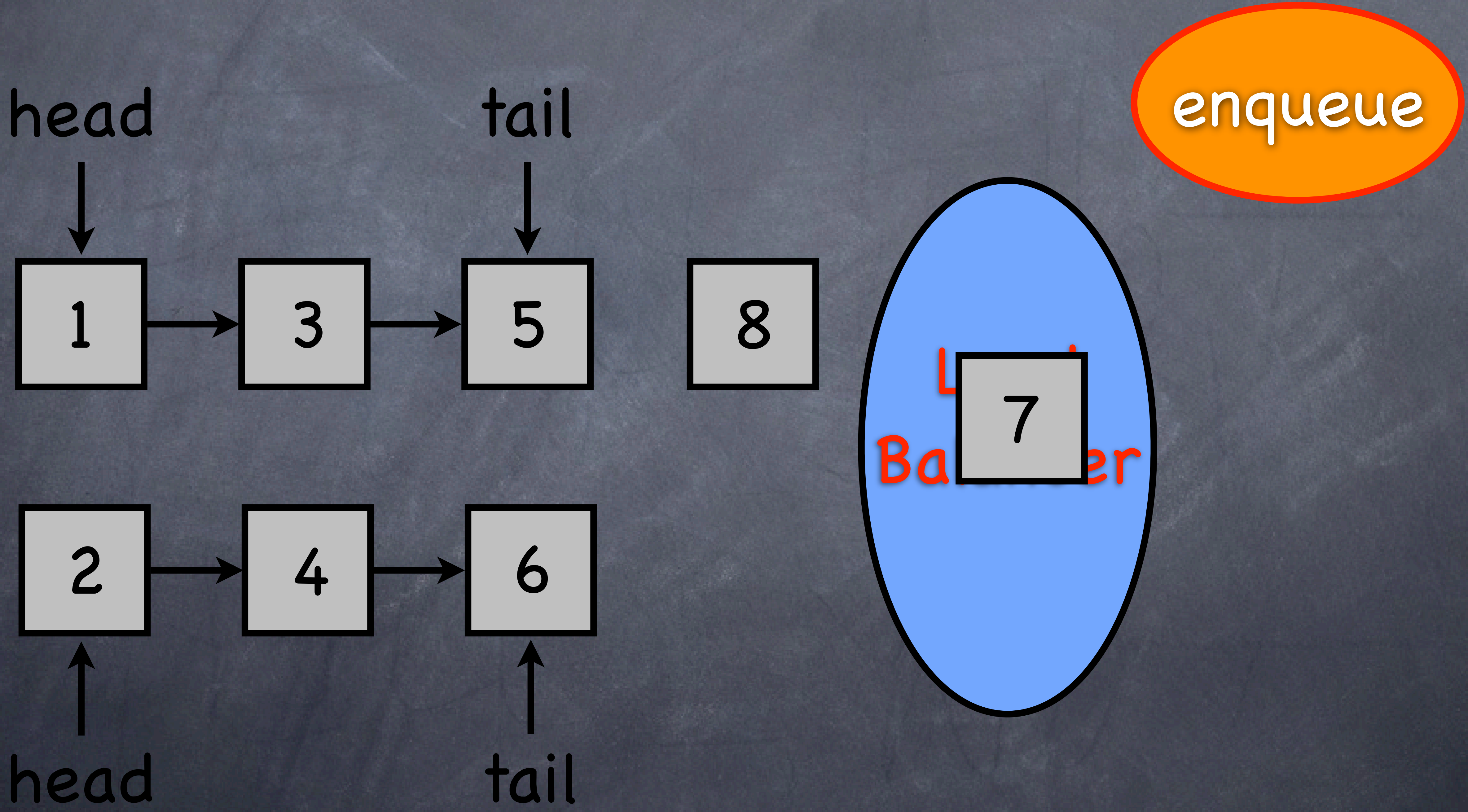
Scal Queue: Load Balancing



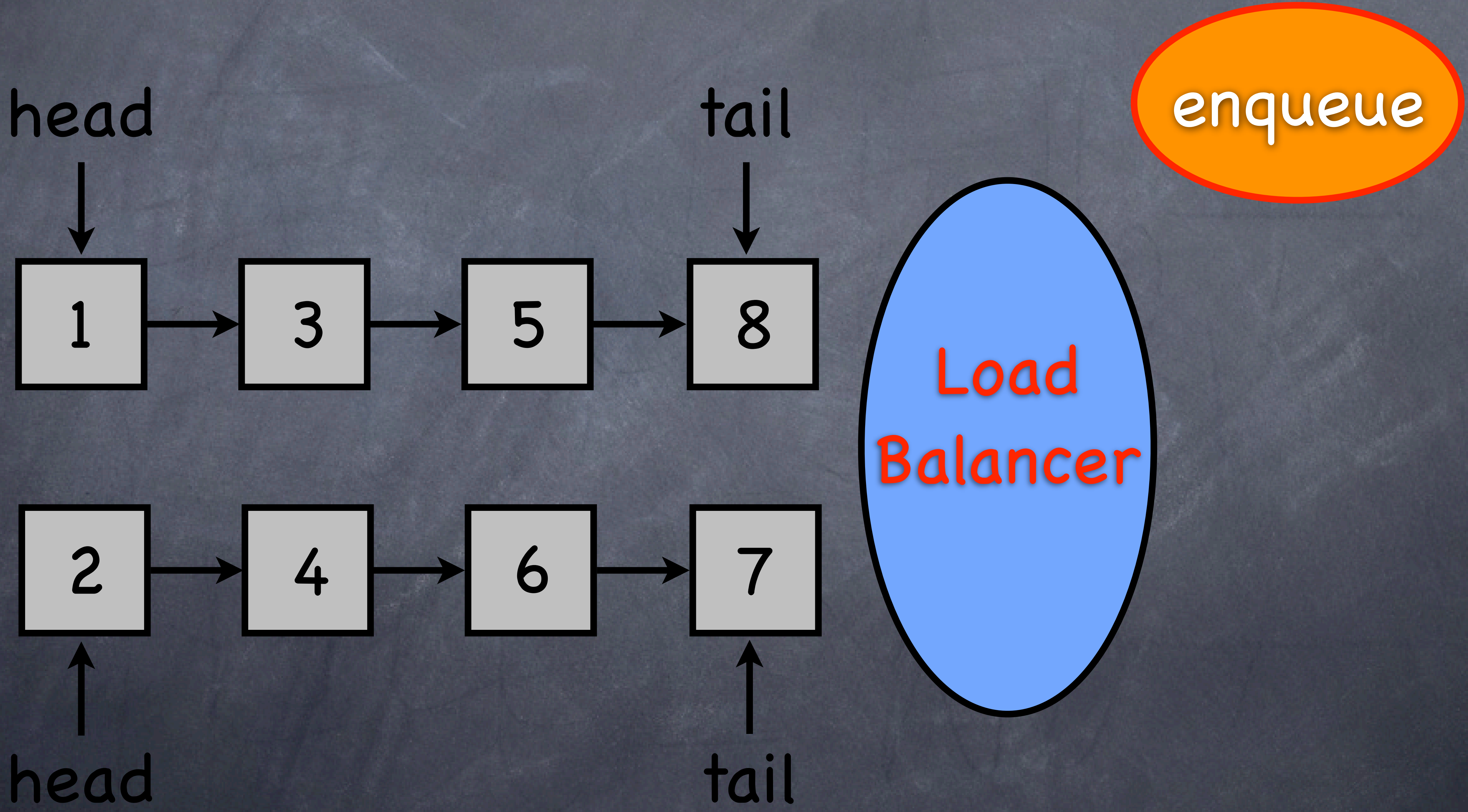
Scal Queue: Load Balancing



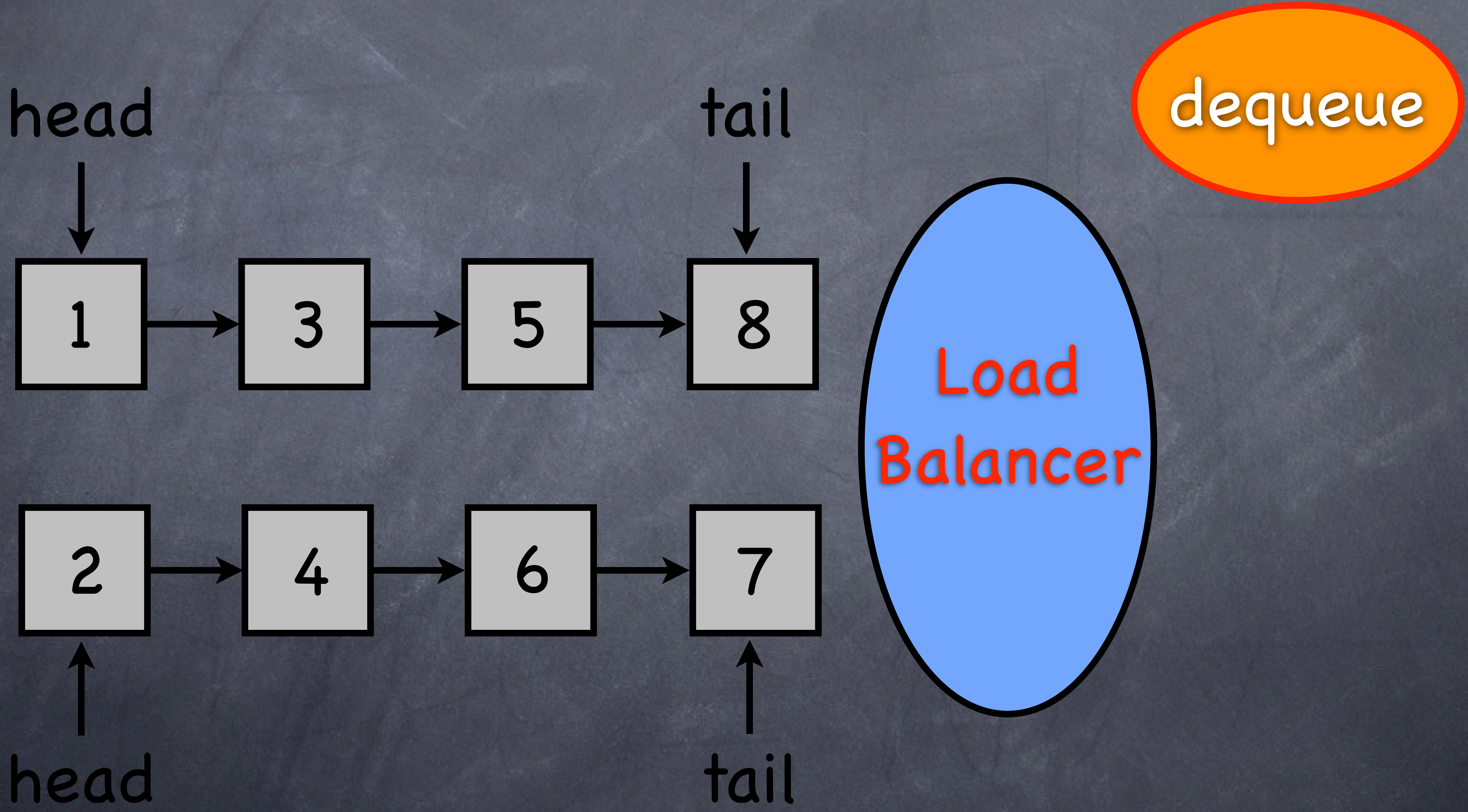
Scal Queue: Load Balancing



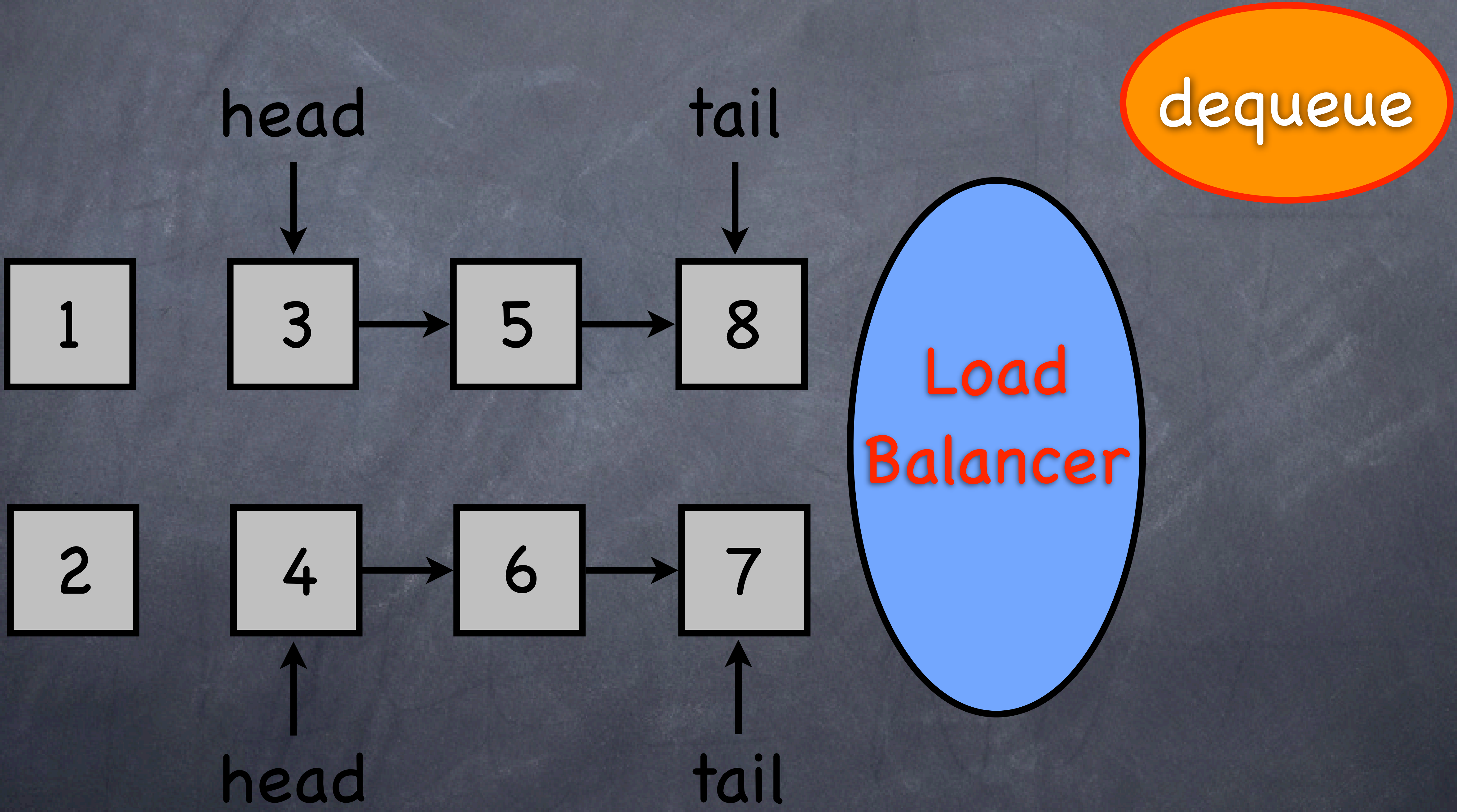
Scal Queue: Load Balancing



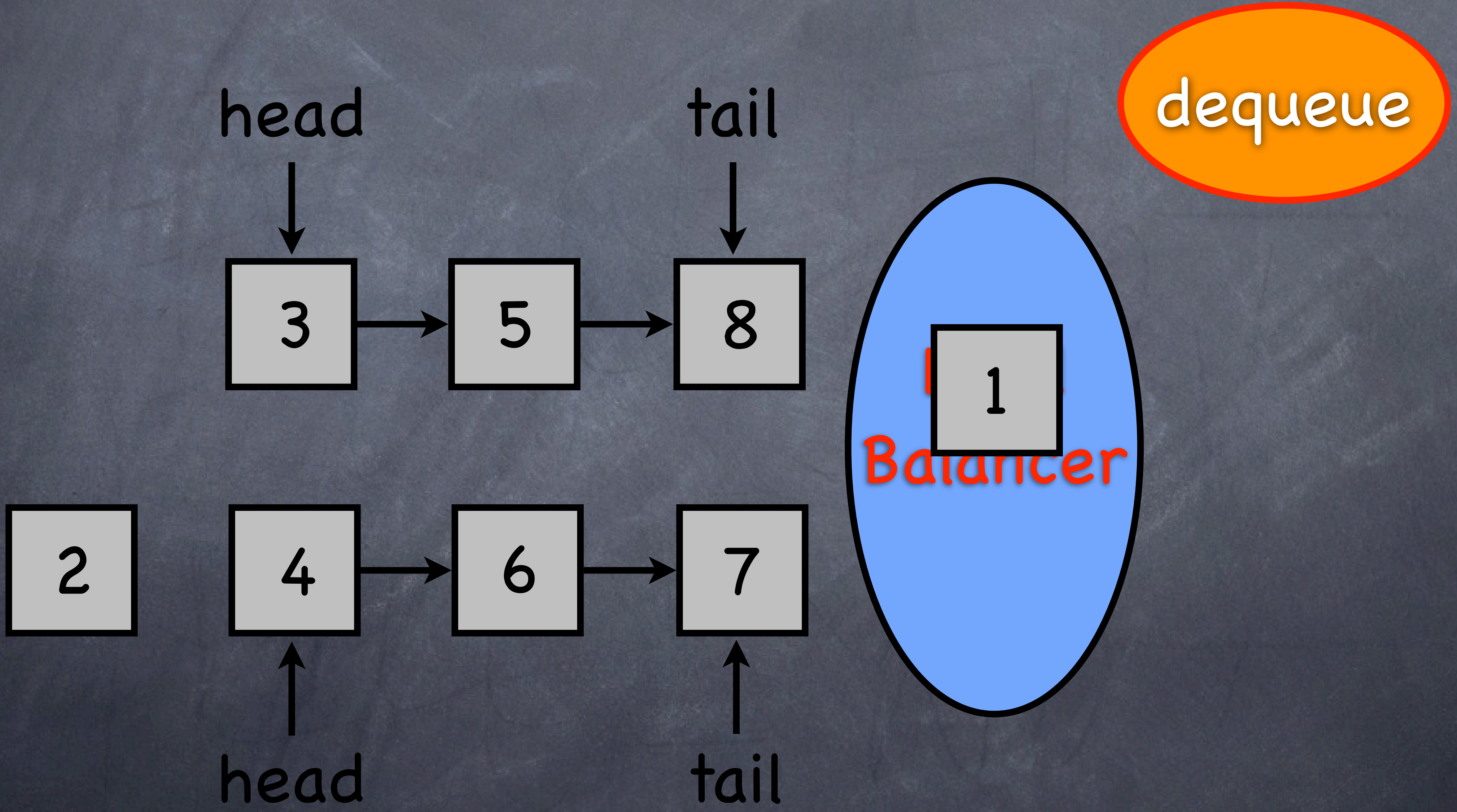
Scal Queue: Load Balancing



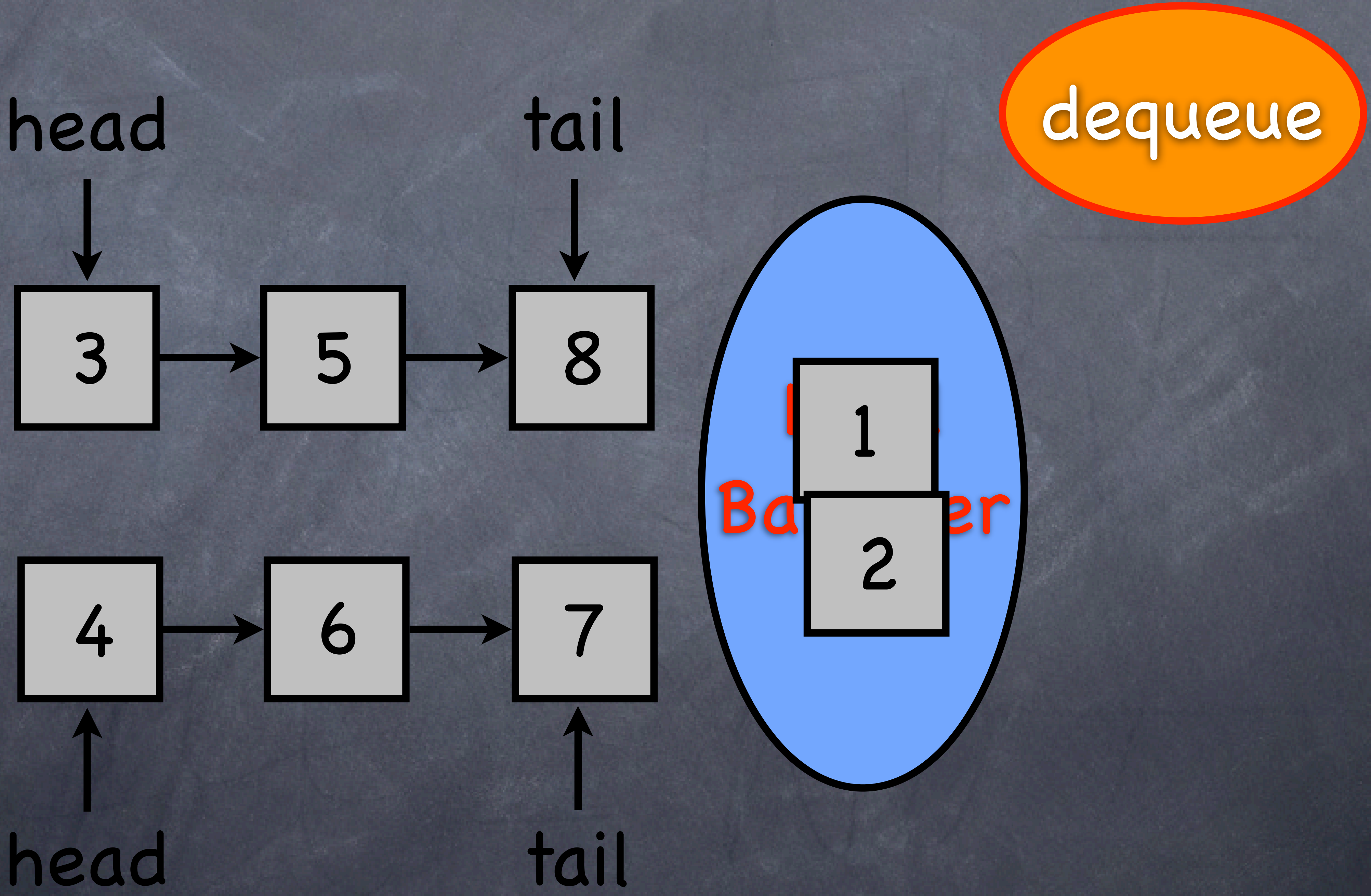
Scal Queue: Load Balancing



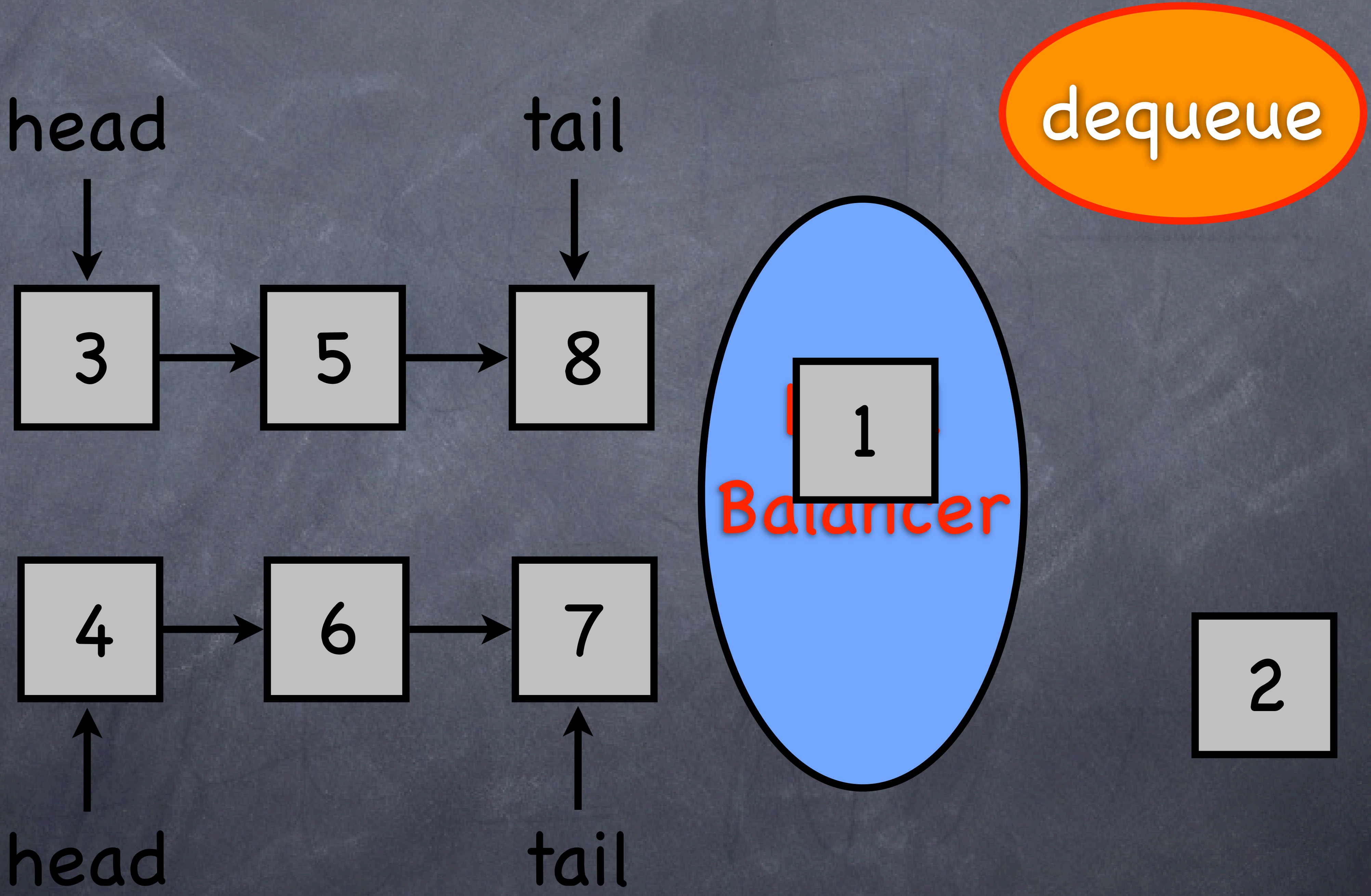
Scal Queue: Load Balancing



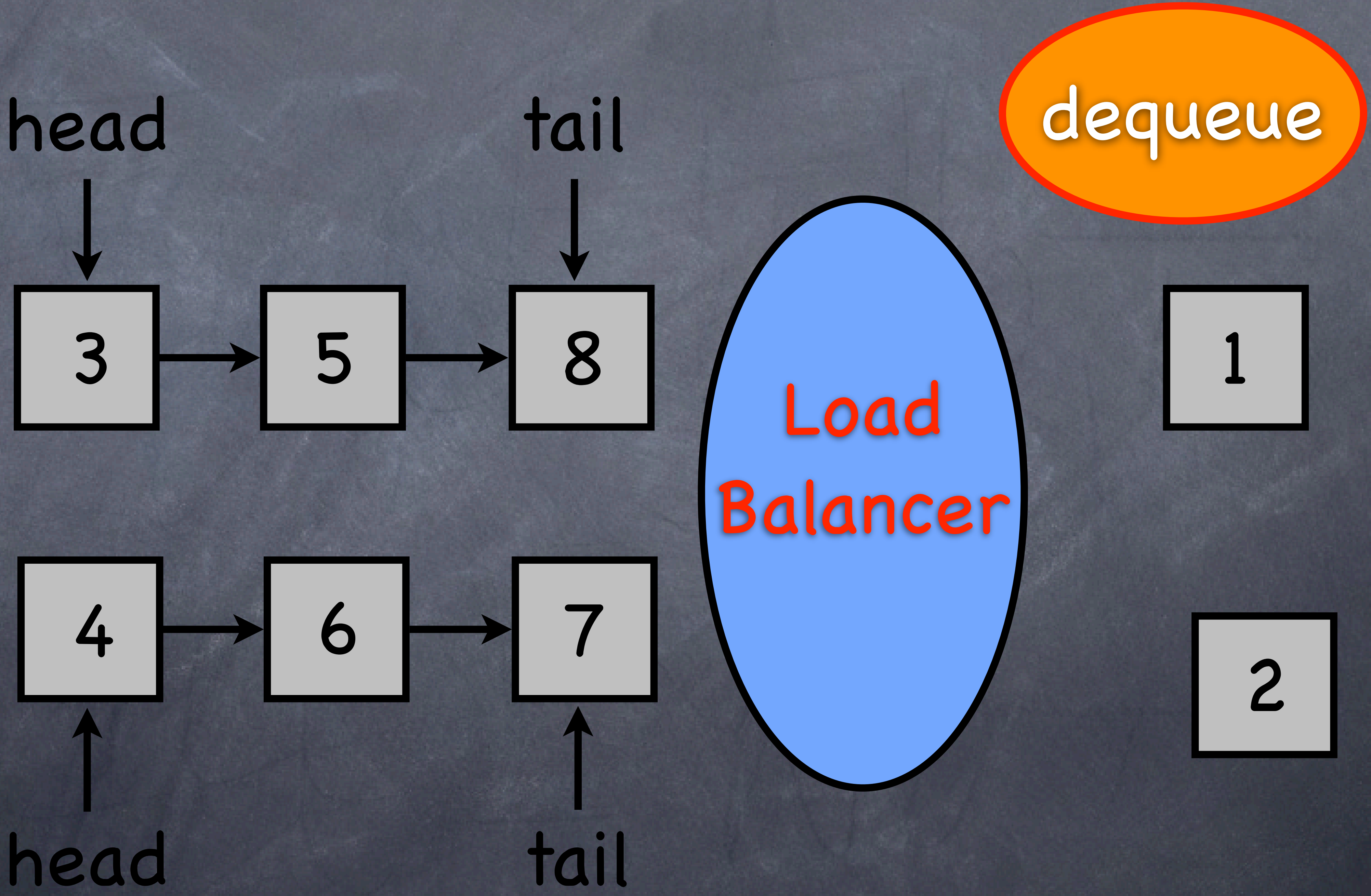
Scal Queue: Load Balancing



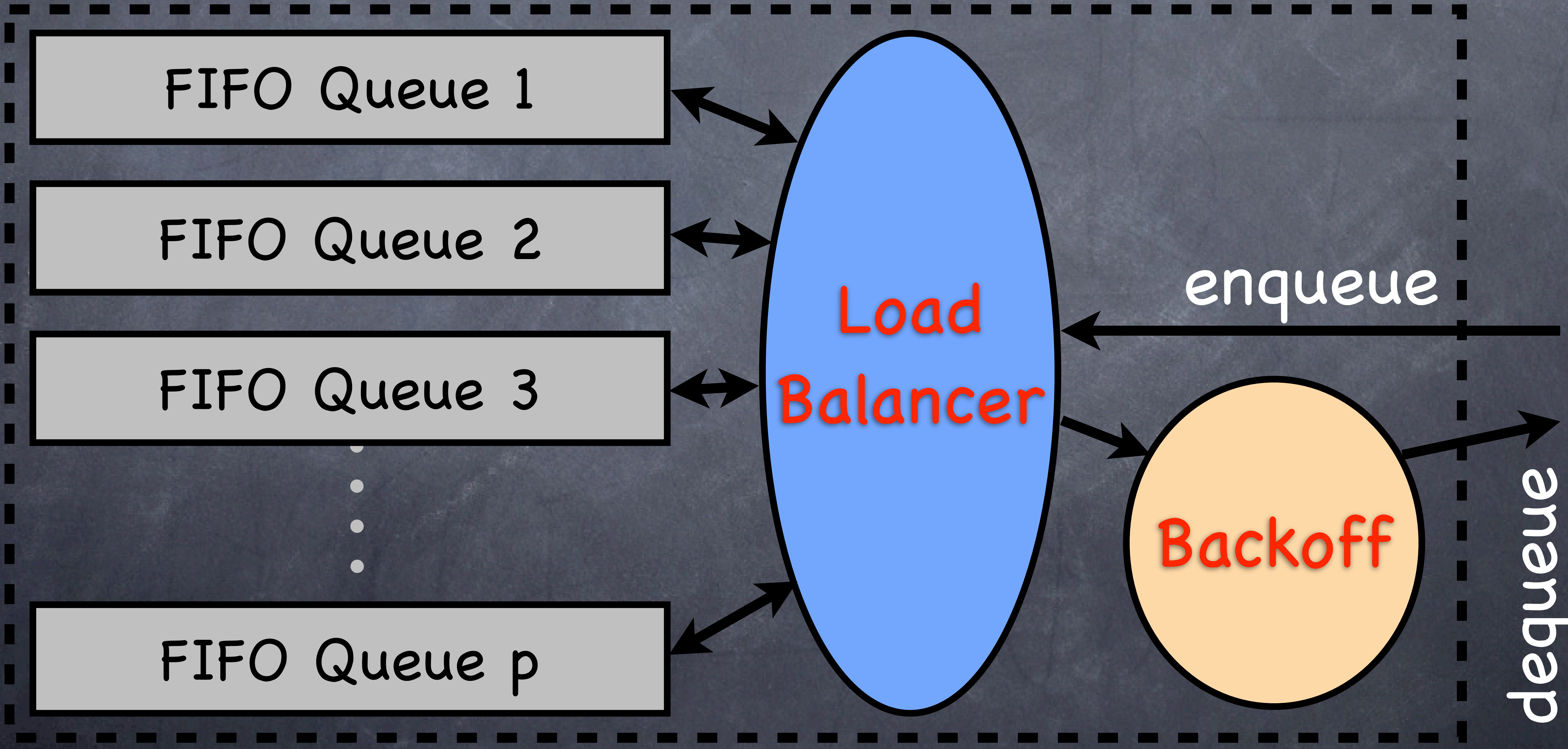
Scal Queue:
Load Balancing



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 · (p-1)	t · (p-1)
Round-Robin Backoff (RR-B)	t · (p-1)	0
Random (	· R · (p-1)	2 · R · (p-1)
Random	p-1)	0
2-R	1)	2 · Q · (p-1)
2-Random	-1)	0
Hierarchical 2-R	Q · (p-1)	2 · Q · (p-1)
Hierarchical 2-Random Backoff (H-2RA-B)	2 · Q · (p-1)	0

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

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

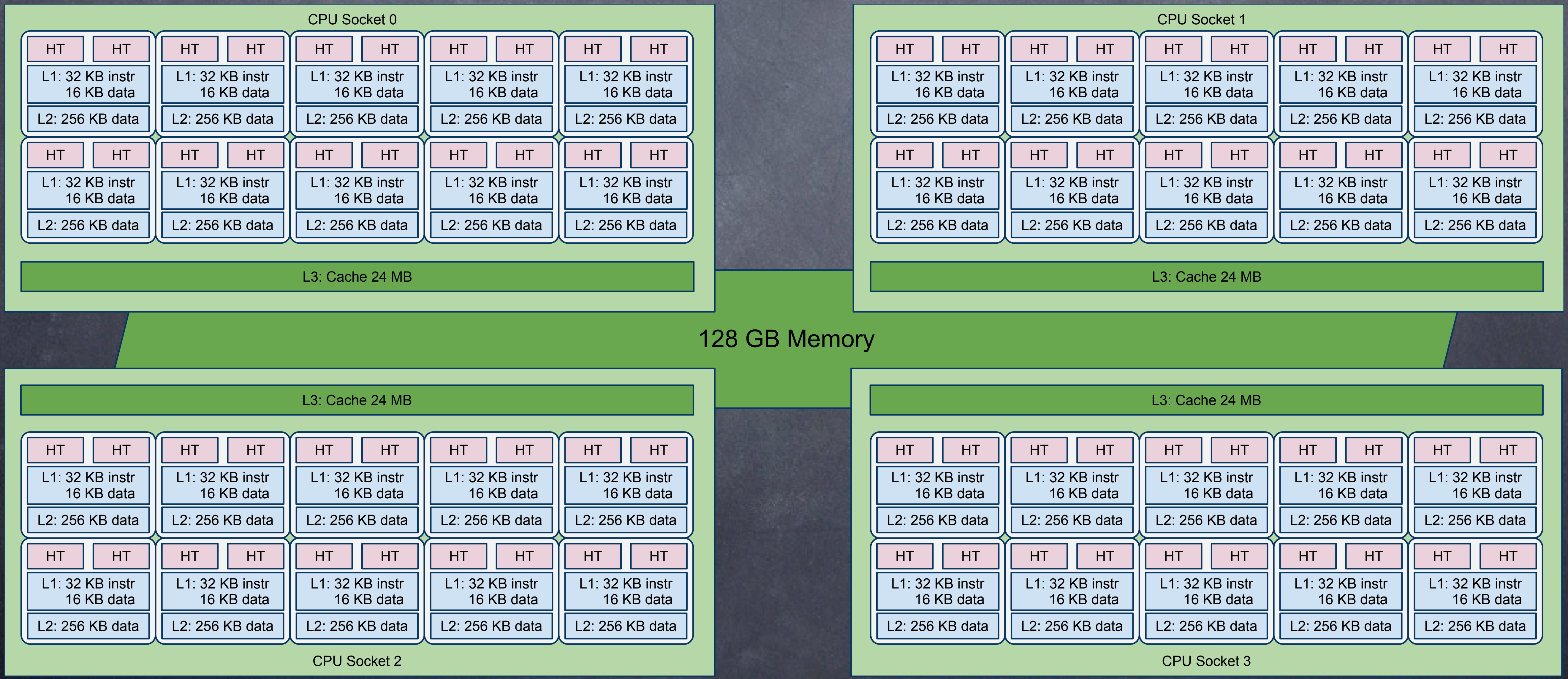
WCSD of Scal Queues

Load b	k	o
Rour	)	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



CPU Socket 0

HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data
HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data

L3: Cache 24 MB

CPU Socket 1

HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data
HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data

L3: Cache 24 MB

128 GB Memory

L3: Cache 24 MB

HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data
HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data

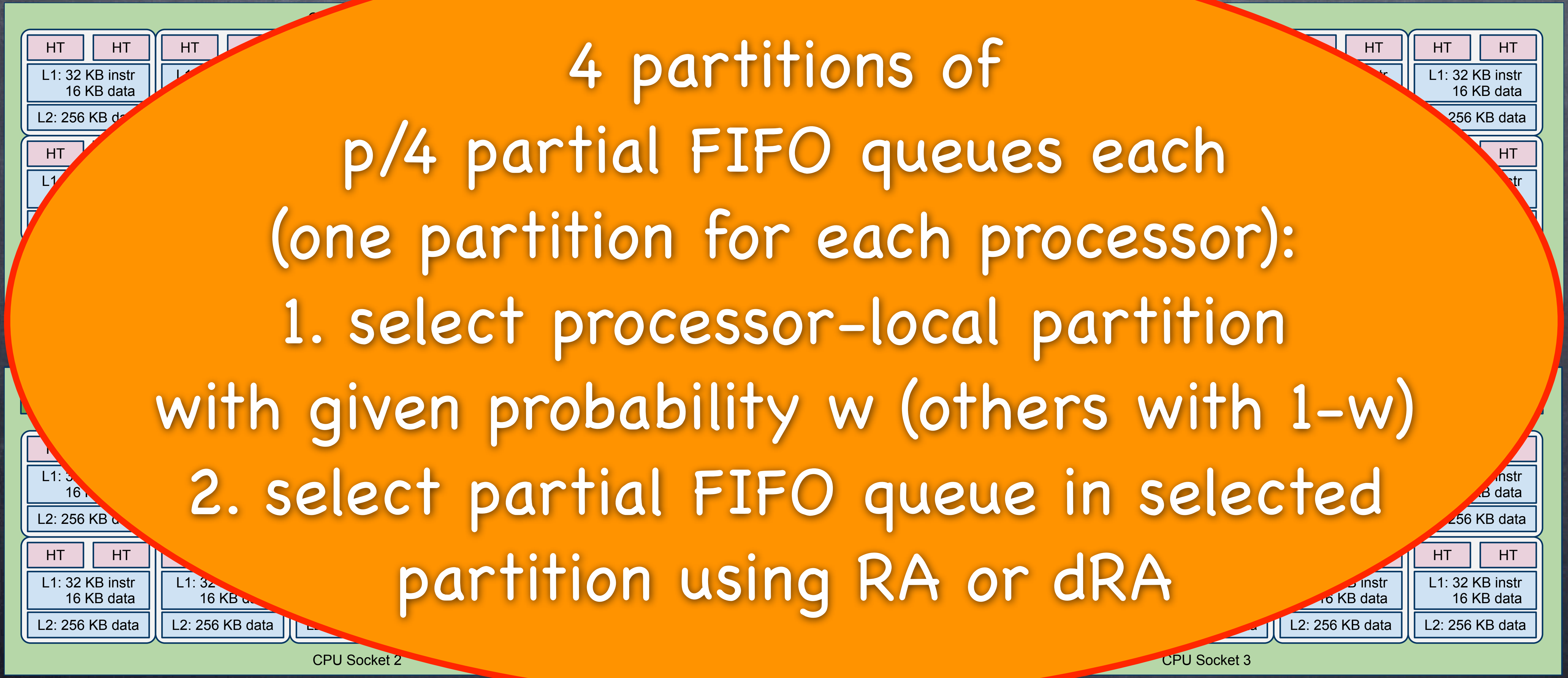
CPU Socket 2

L3: Cache 24 MB

HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data
HT HT	HT HT	HT HT	HT HT	HT HT
L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data	L1: 32 KB instr 16 KB data
L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data	L2: 256 KB data

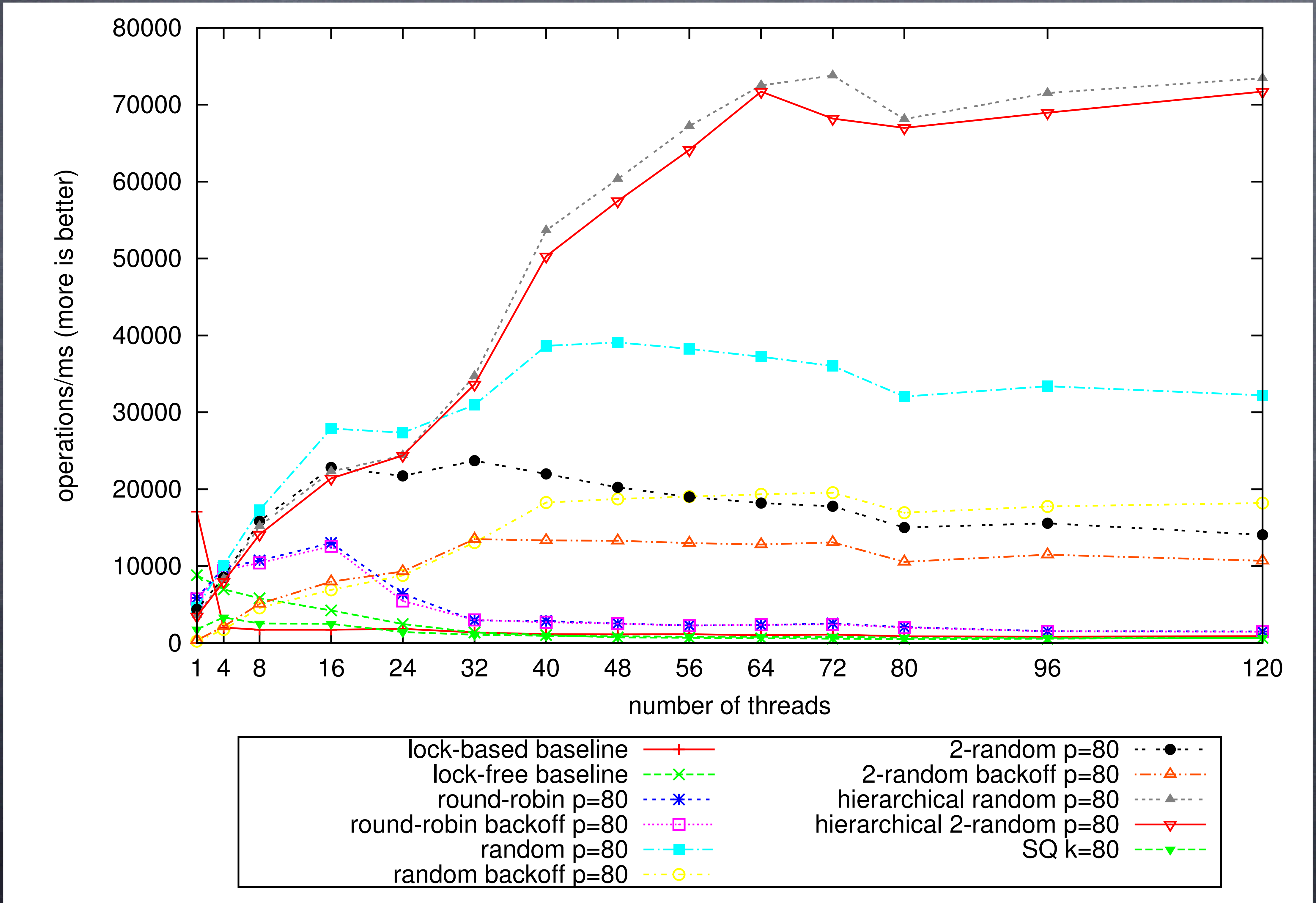
CPU Socket 3

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



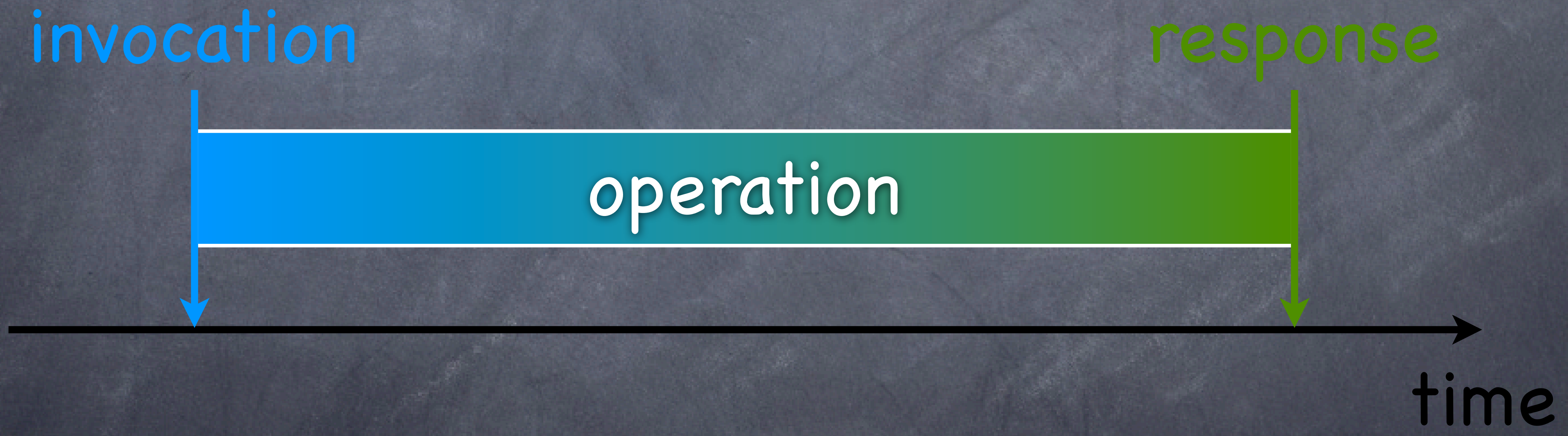
4 partitions of
p/4 partial FIFO queues each
(one partition for each processor):
1. select processor-local partition
with given probability w (others with 1-w)
2. select partial FIFO queue in selected
partition using RA or dRA

Performance & Scalability



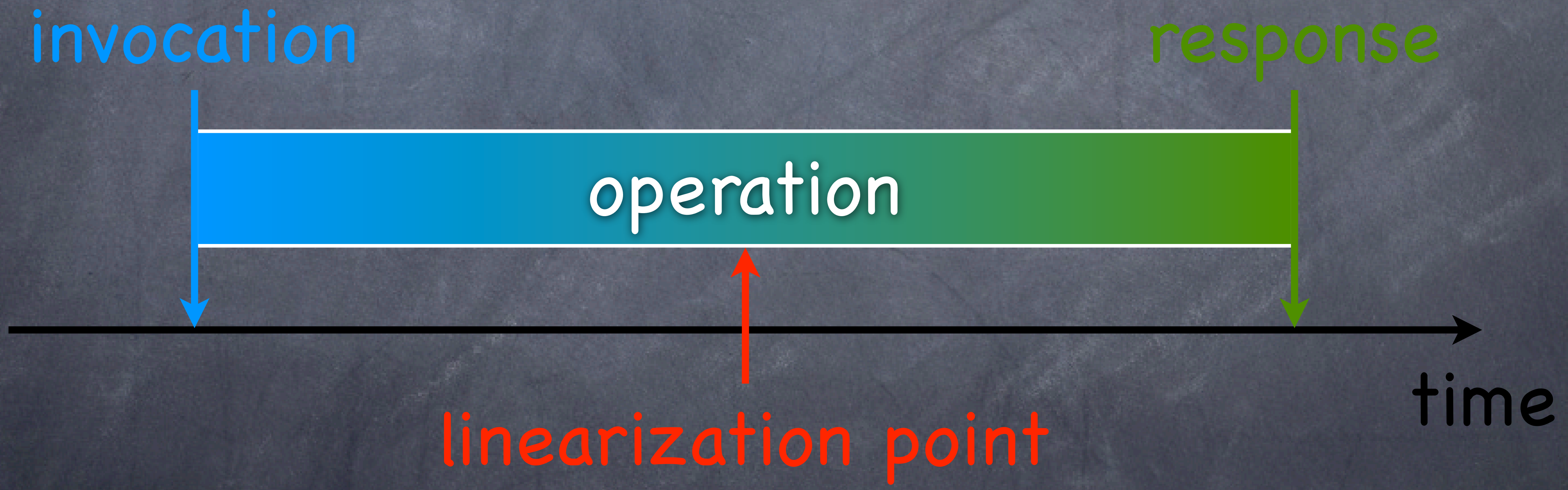
Execution History

Sequence of Time-Stamped Invocation and Response Events

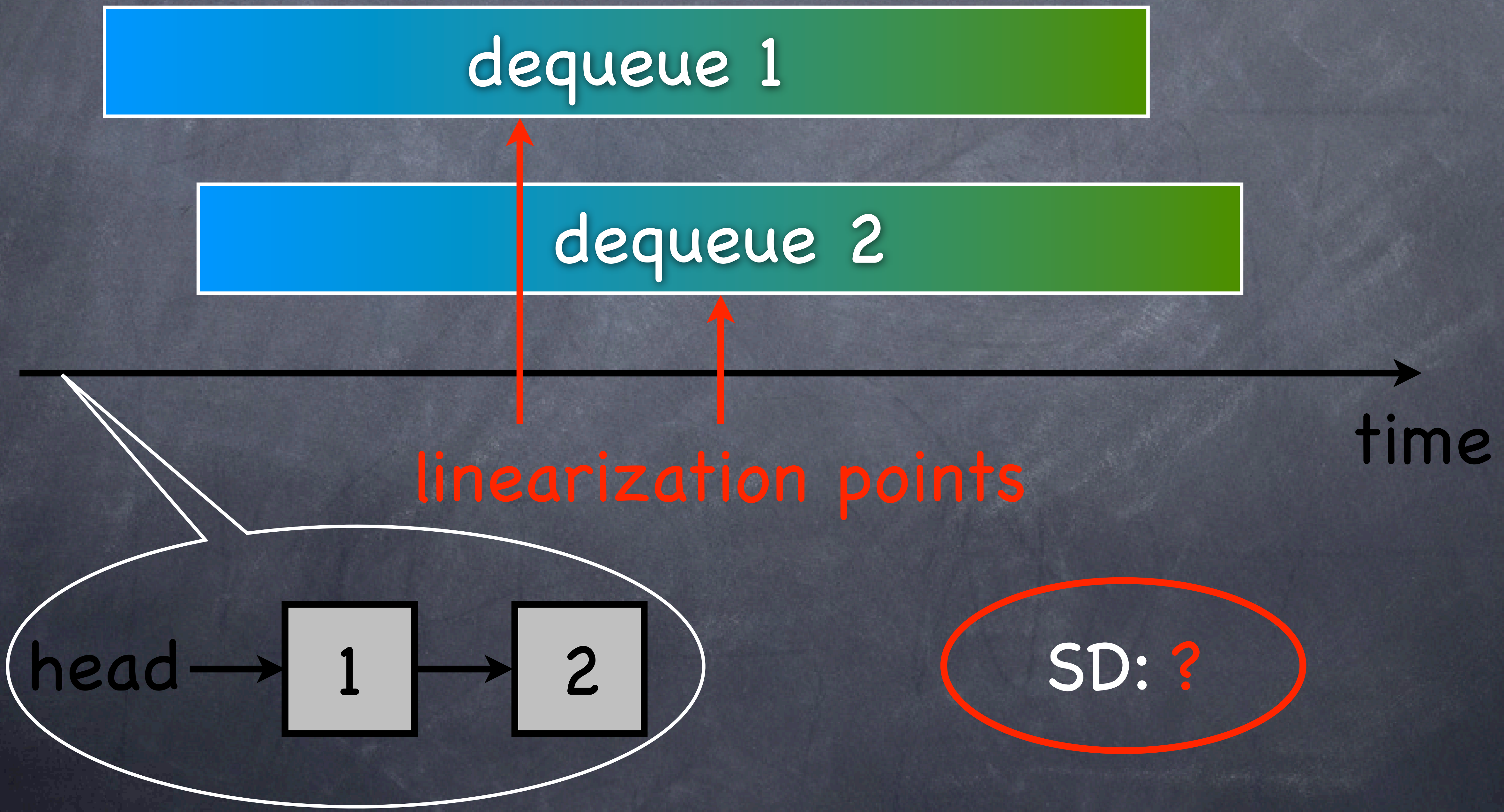


Execution History

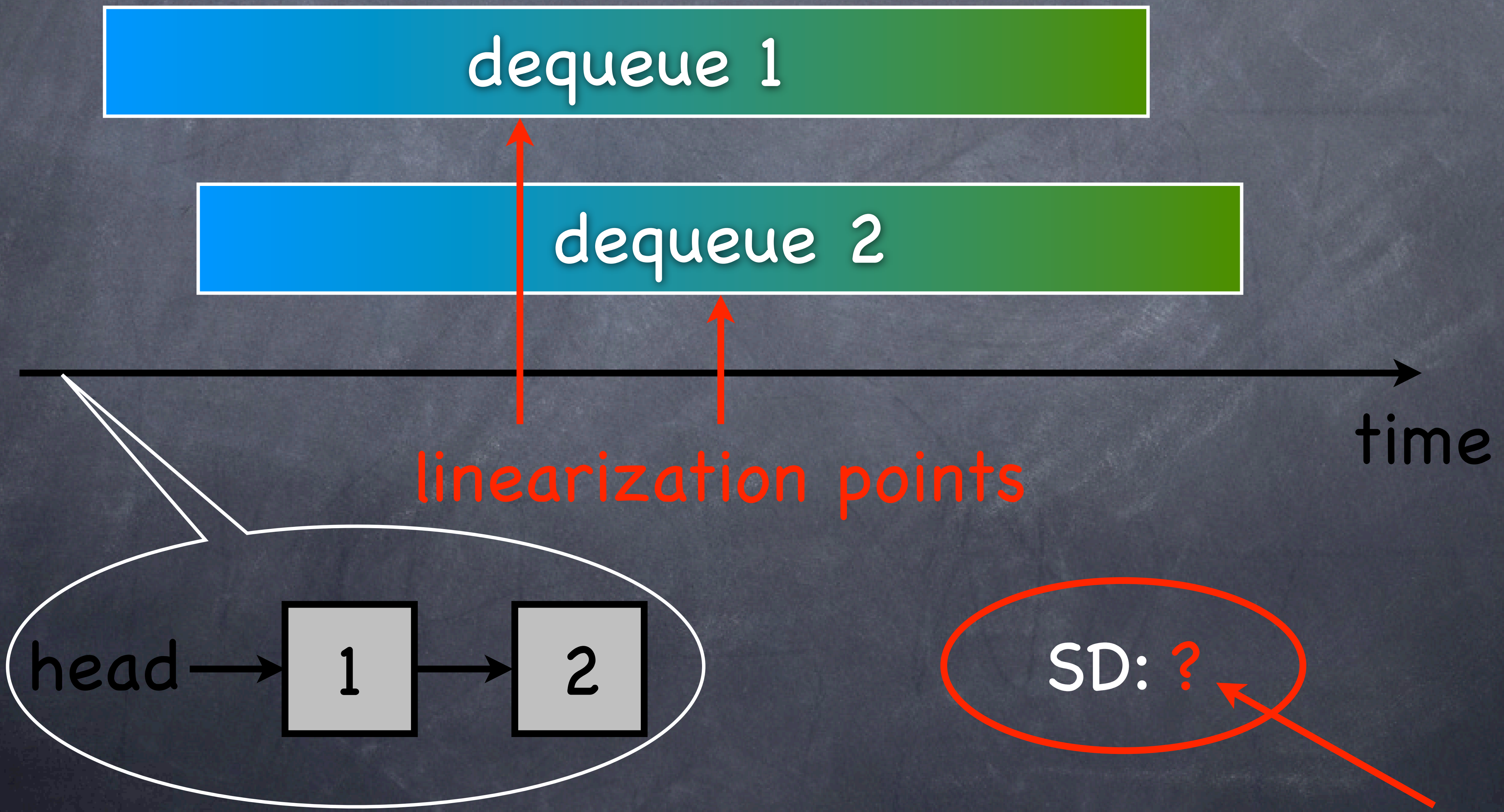
Sequence of Time-Stamped Invocation and Response Events



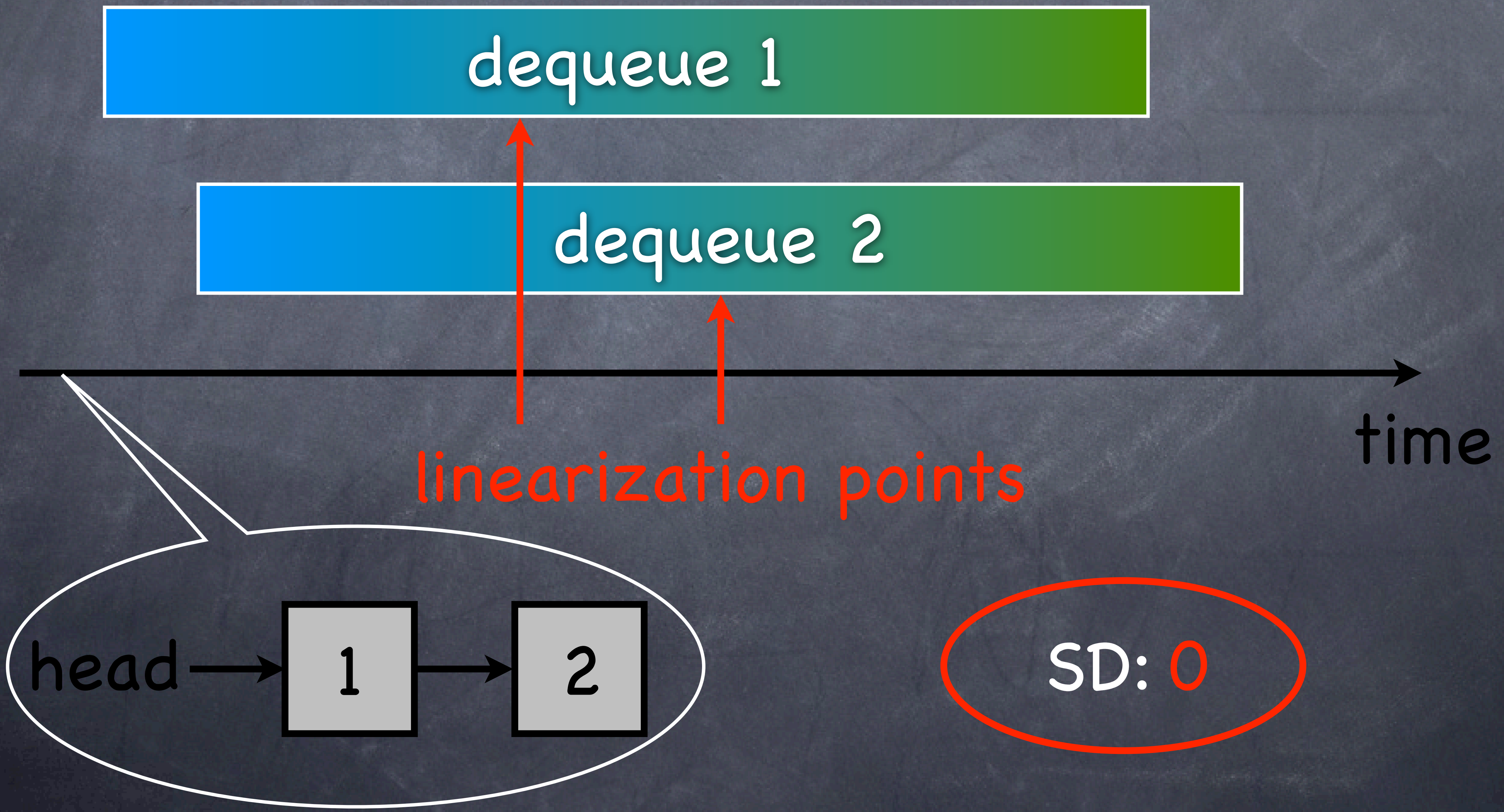
Measuring Semantical Deviation (SD)



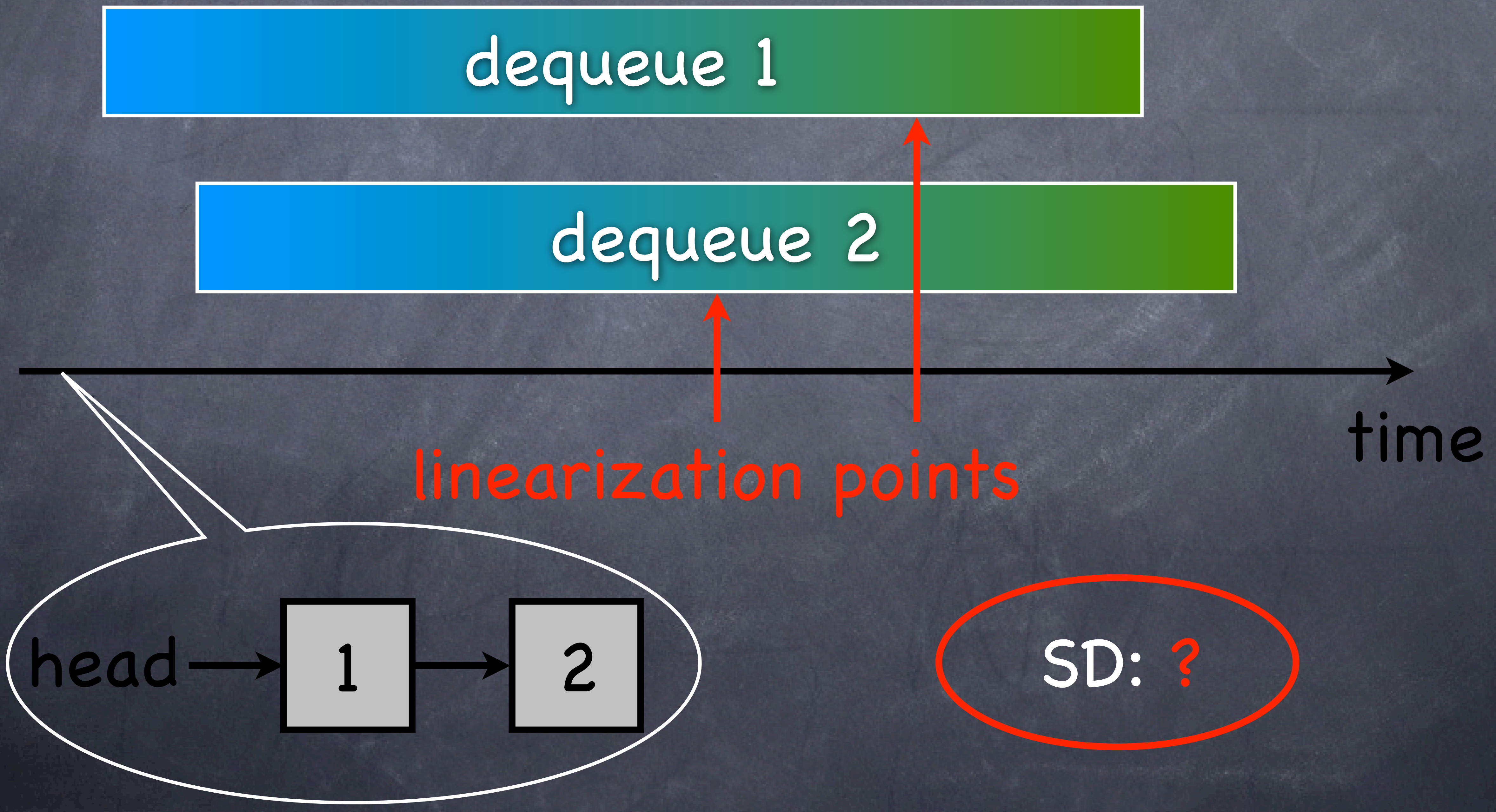
Measuring Semantical Deviation (SD)



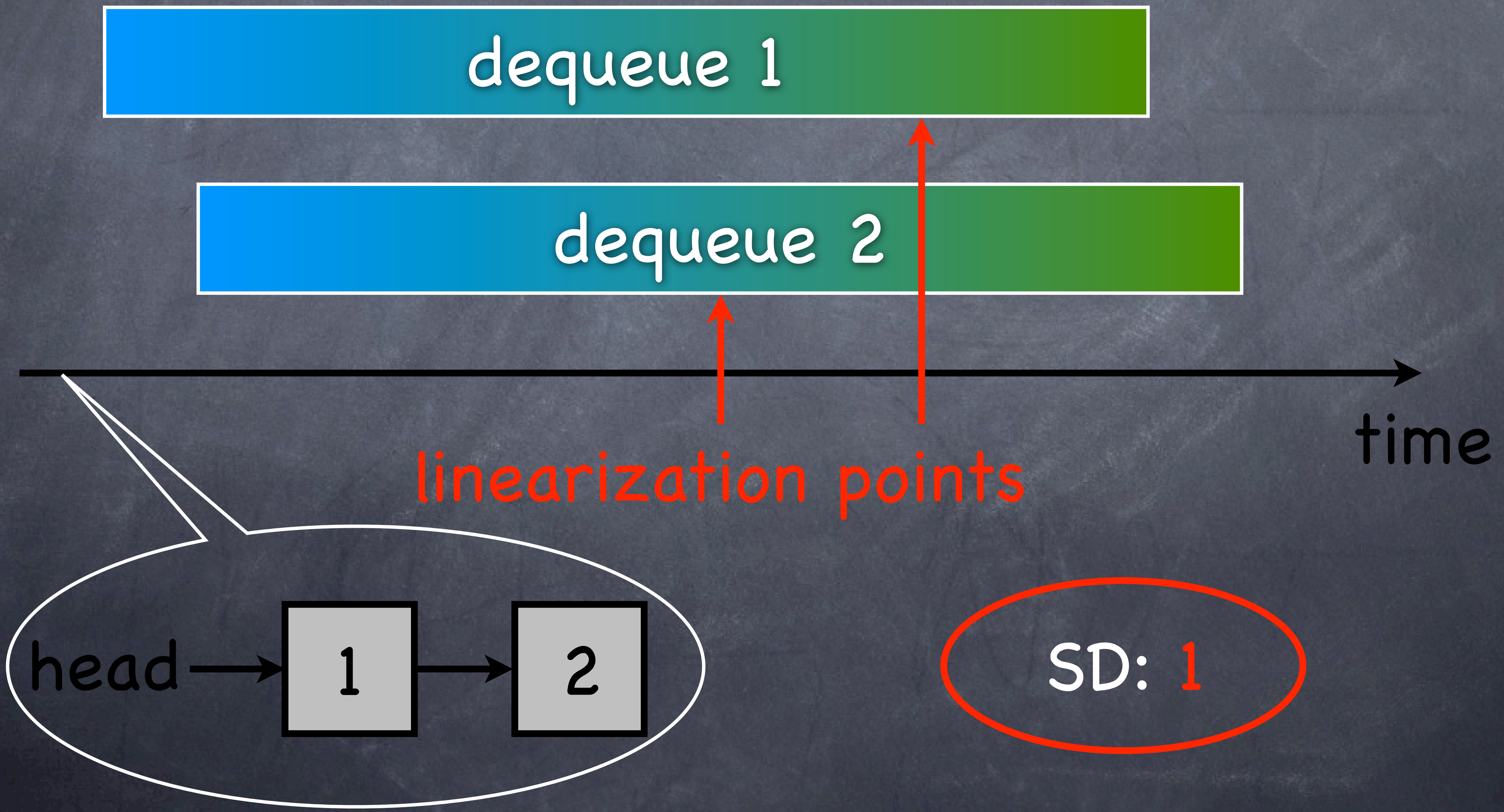
Measuring Semantical Deviation (SD)



Measuring Semantical Deviation (SD)



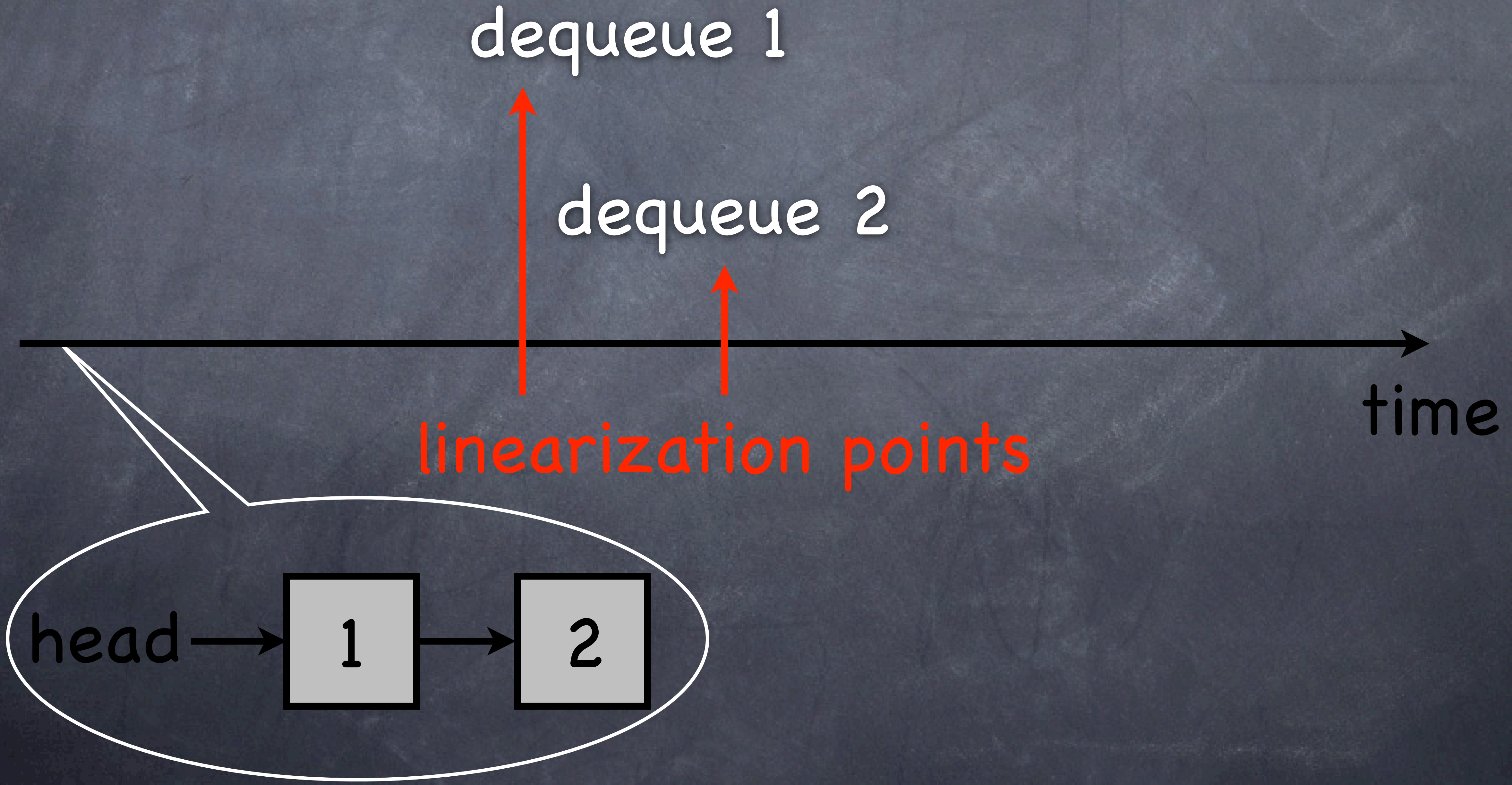
Measuring Semantical Deviation (SD)



SD: 1

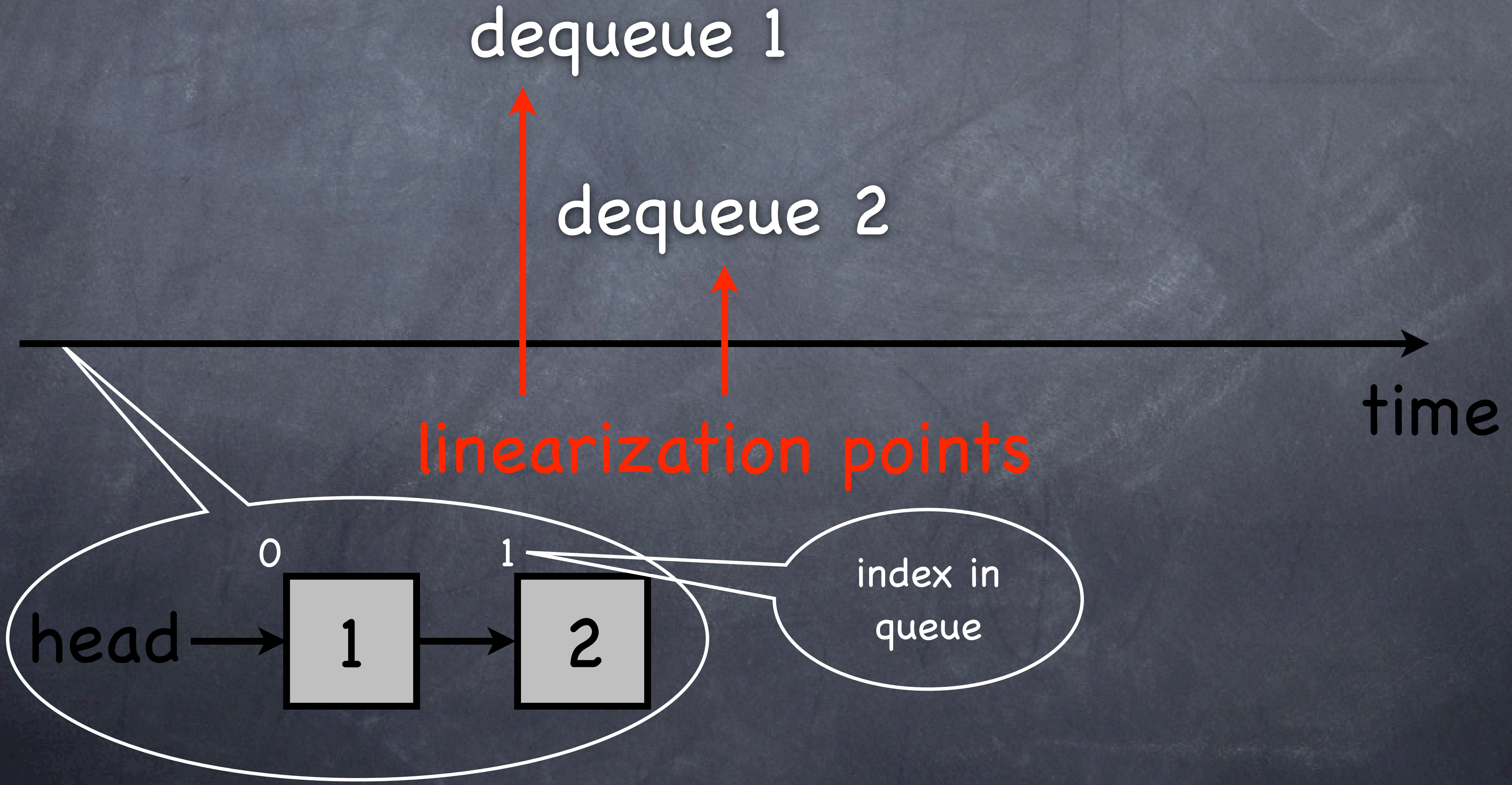
Sequential History

Sequence of Operations (Linearization Points)



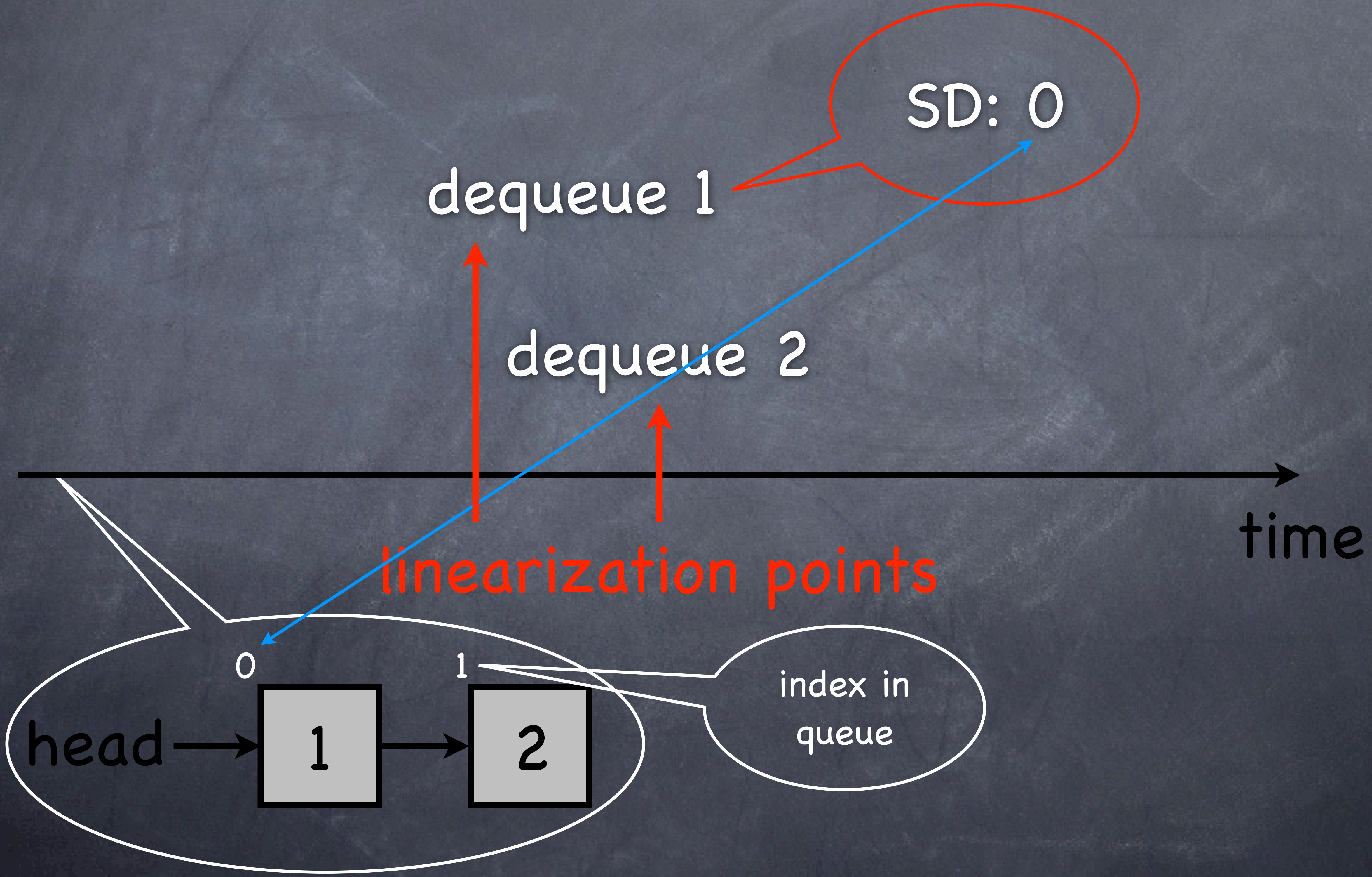
Sequential History

Sequence of Operations (Linearization Points)



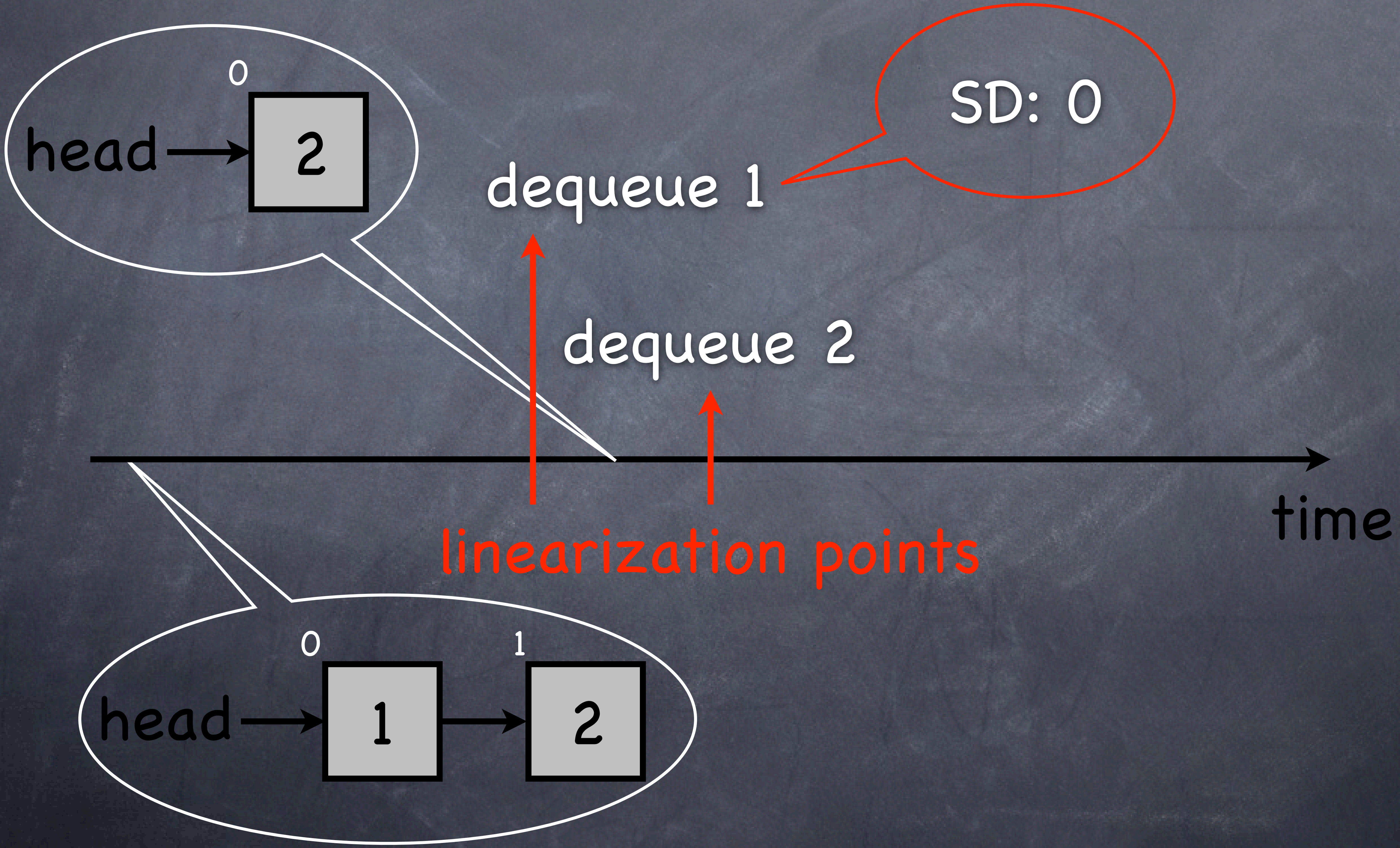
Sequential History

Sequence of Operations (Linearization Points)



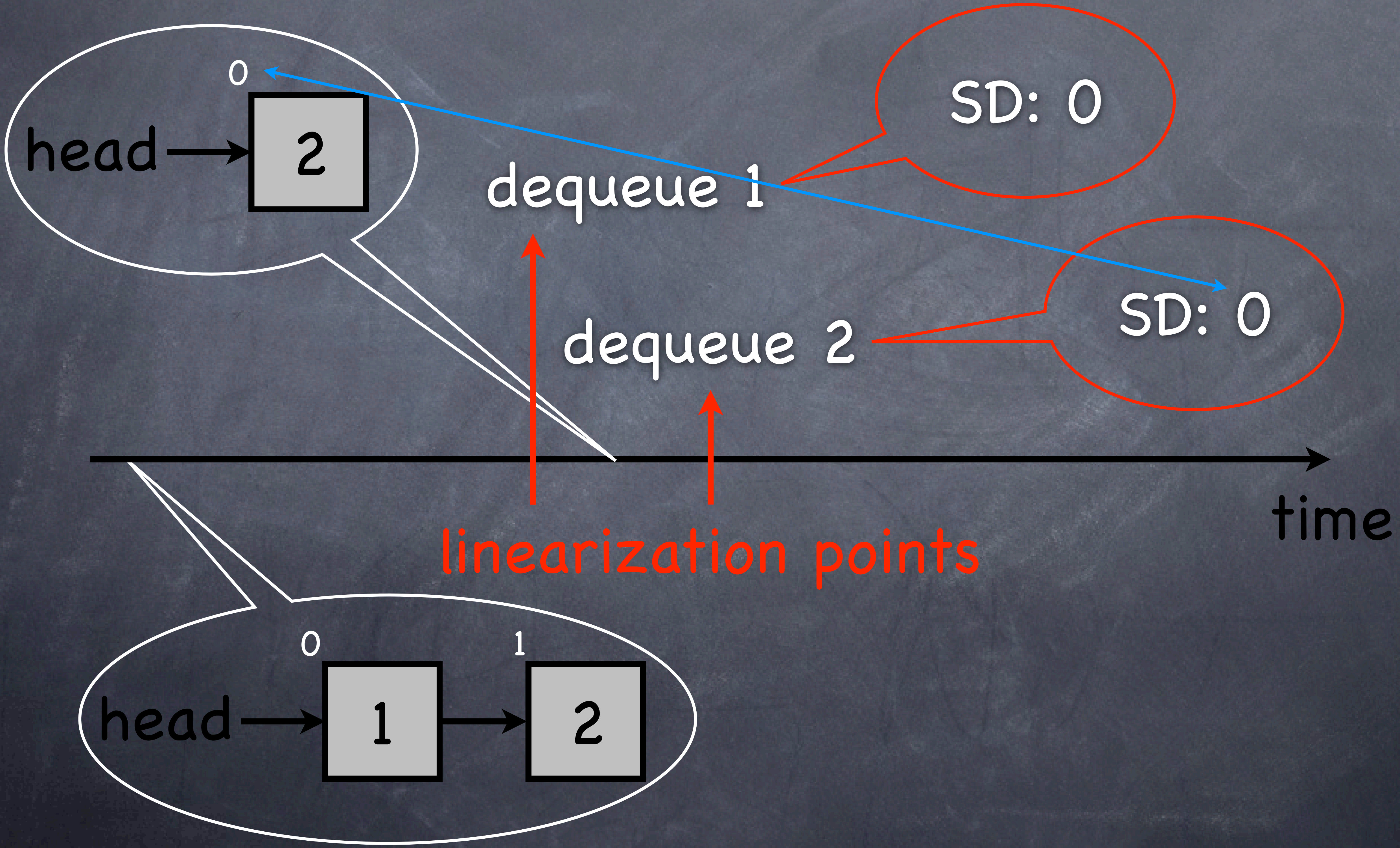
Sequential History

Sequence of Operations (Linearization Points)



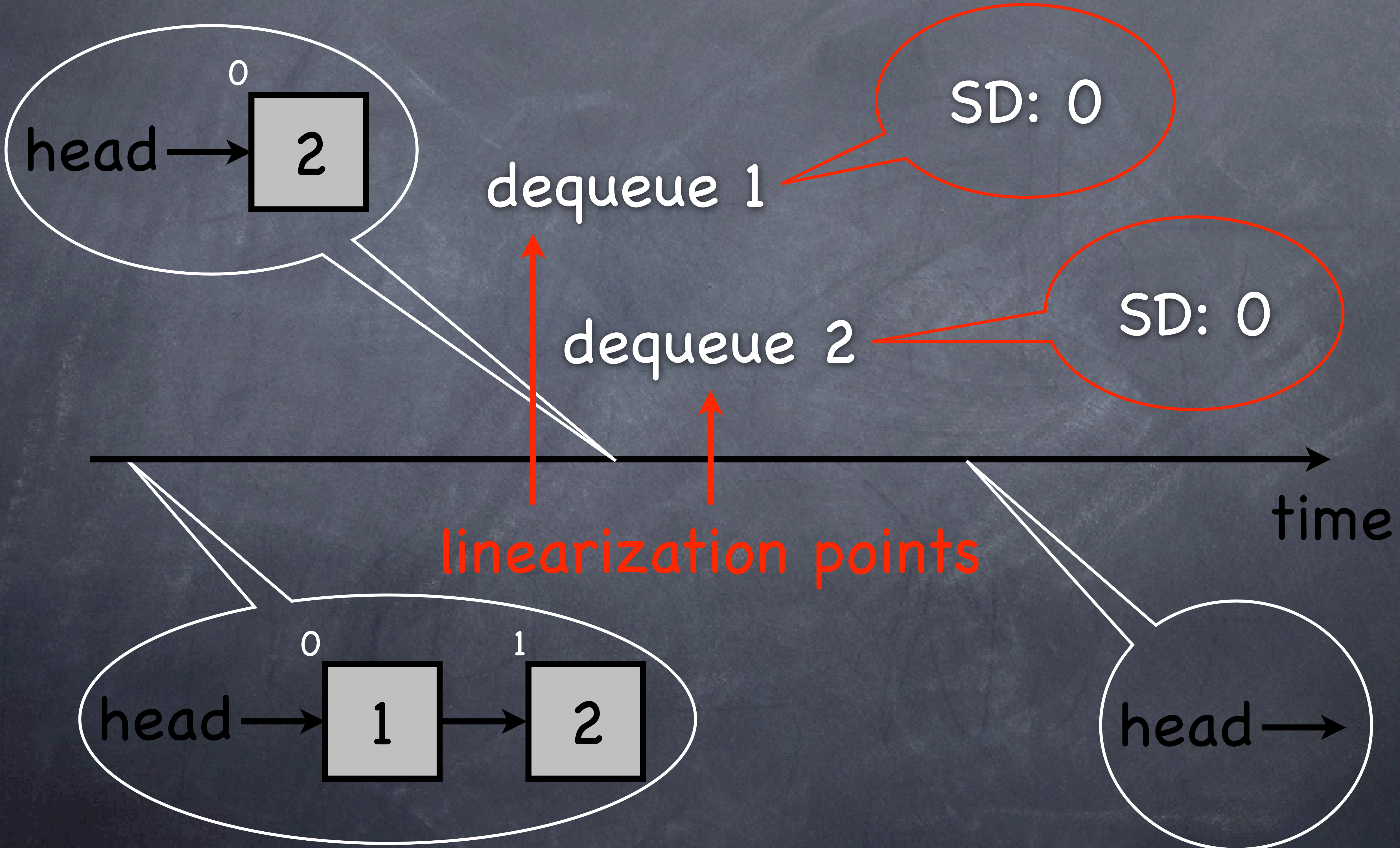
Sequential History

Sequence of Operations (Linearization Points)



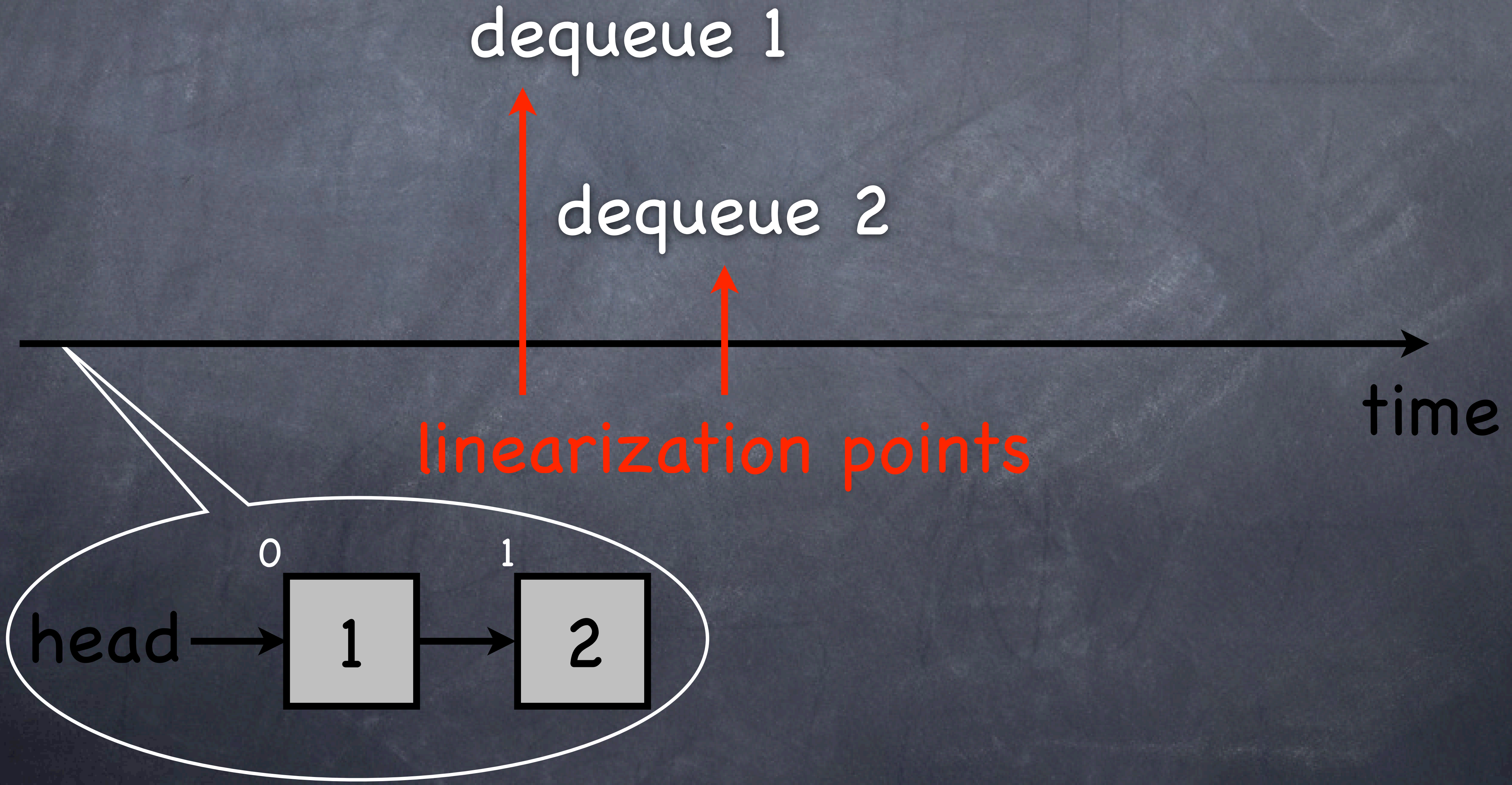
Sequential History

Sequence of Operations (Linearization Points)



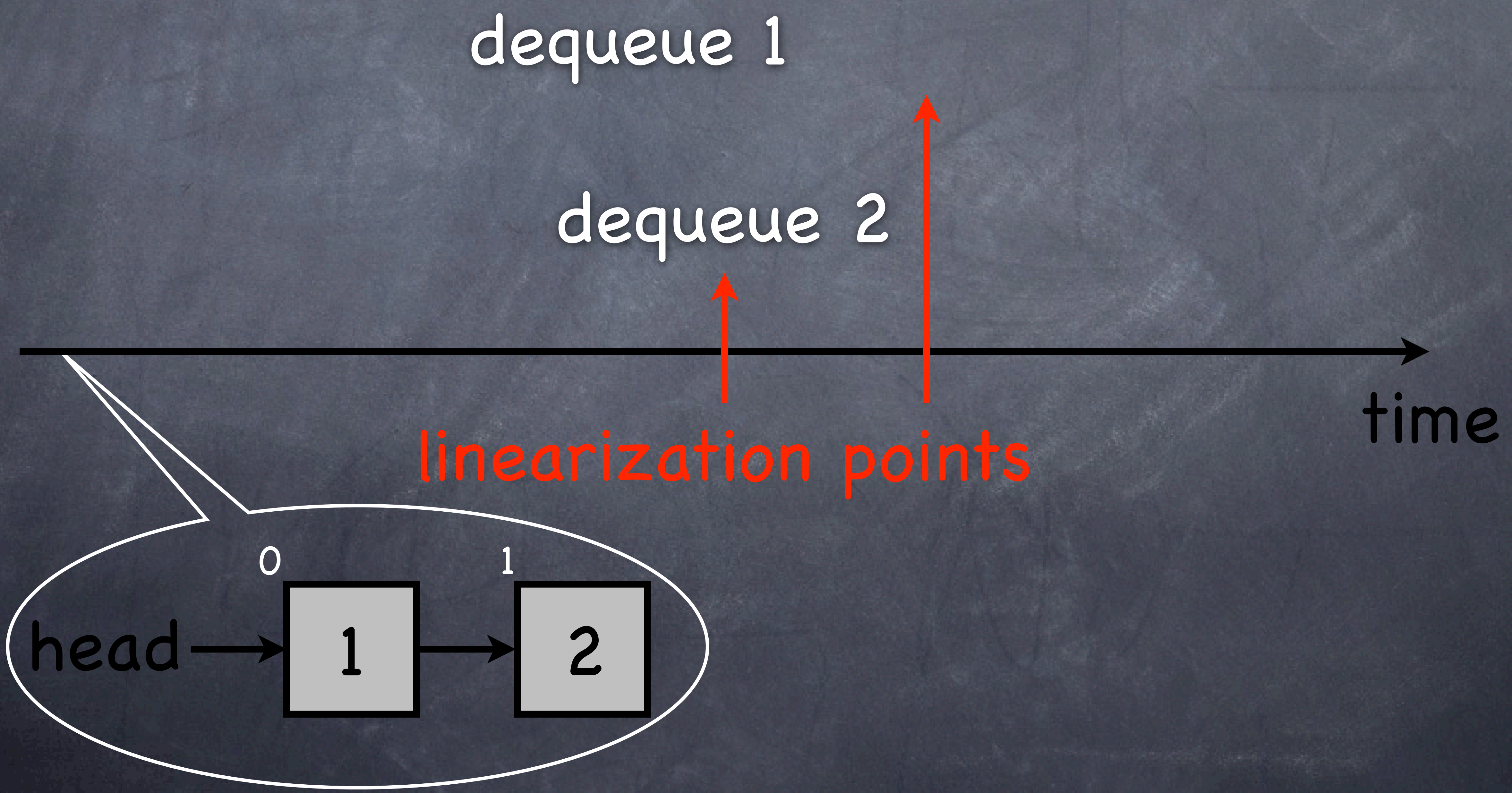
Sequential History II

Sequence of Operations (Linearization Points)



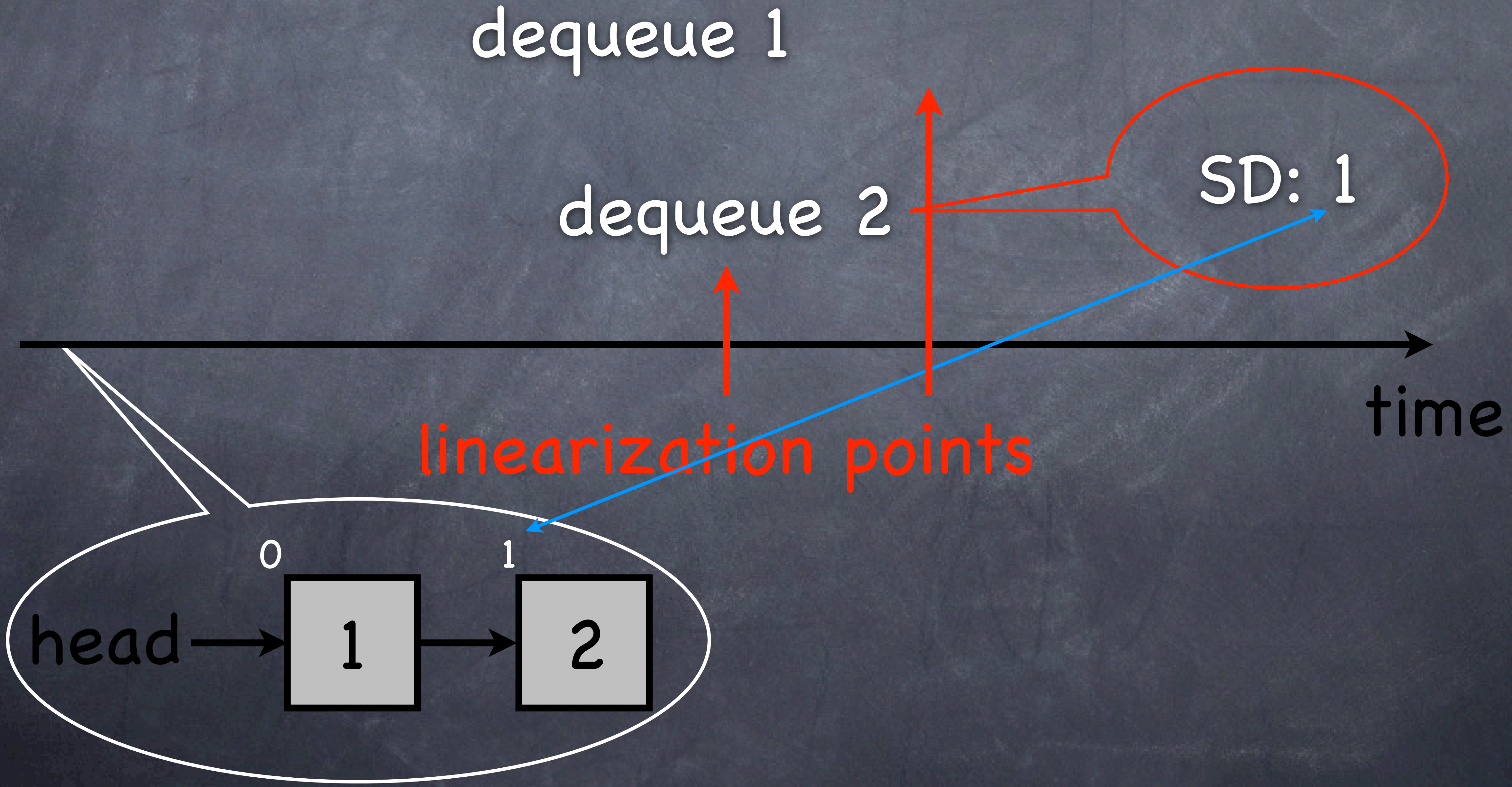
Sequential History II

Sequence of Operations (Linearization Points)



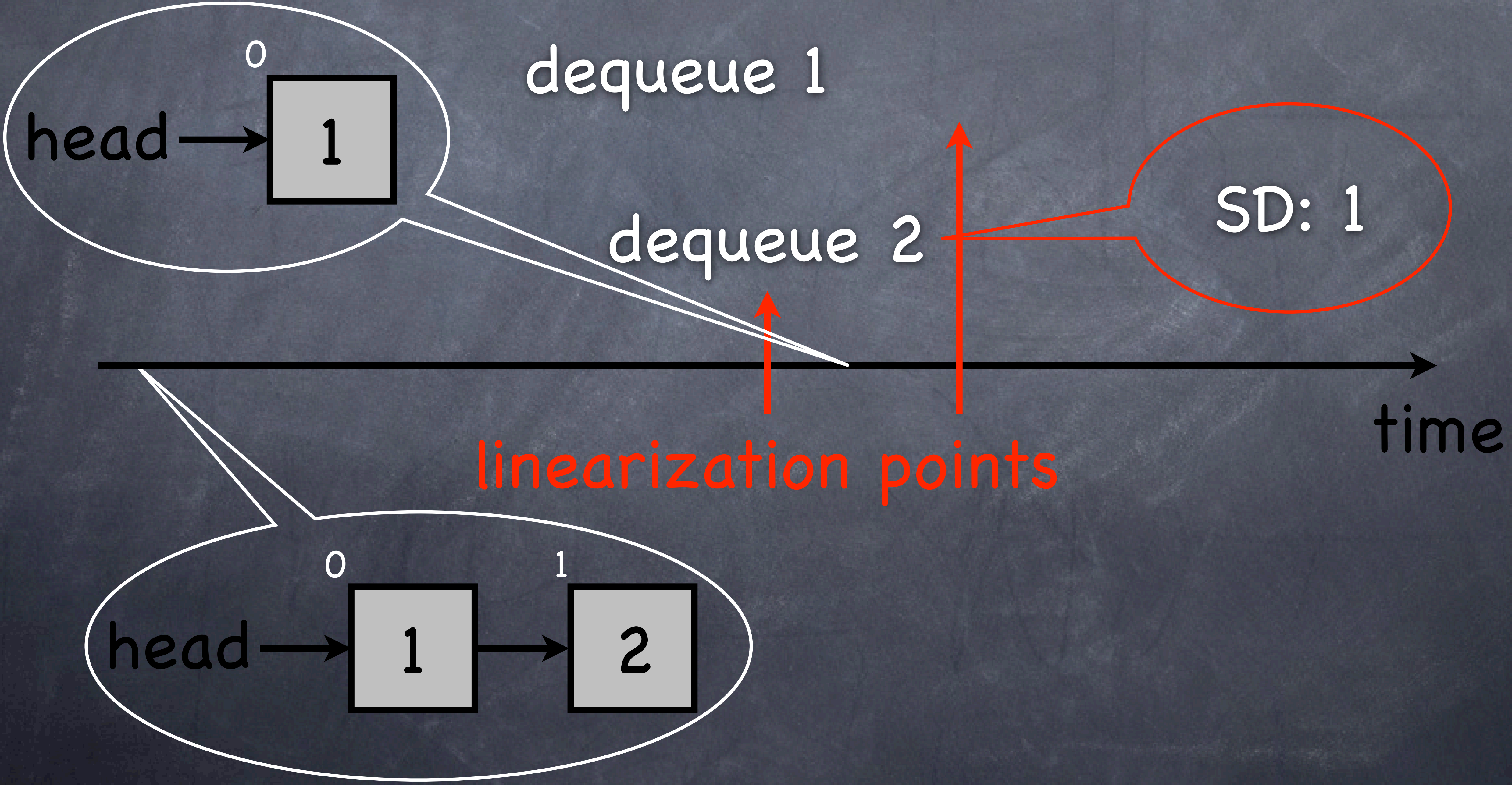
Sequential History II

Sequence of Operations (Linearization Points)



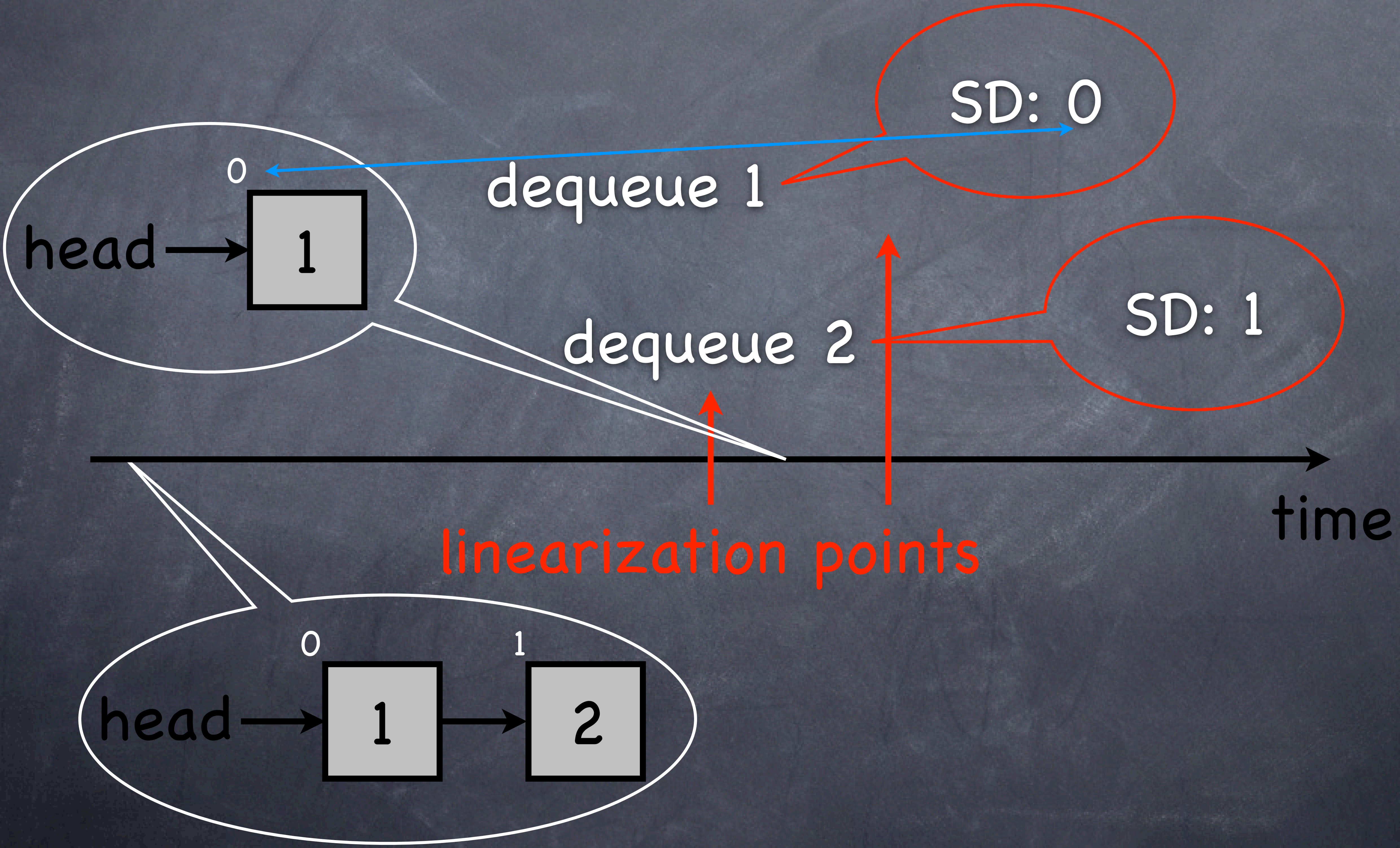
Sequential History II

Sequence of Operations (Linearization Points)



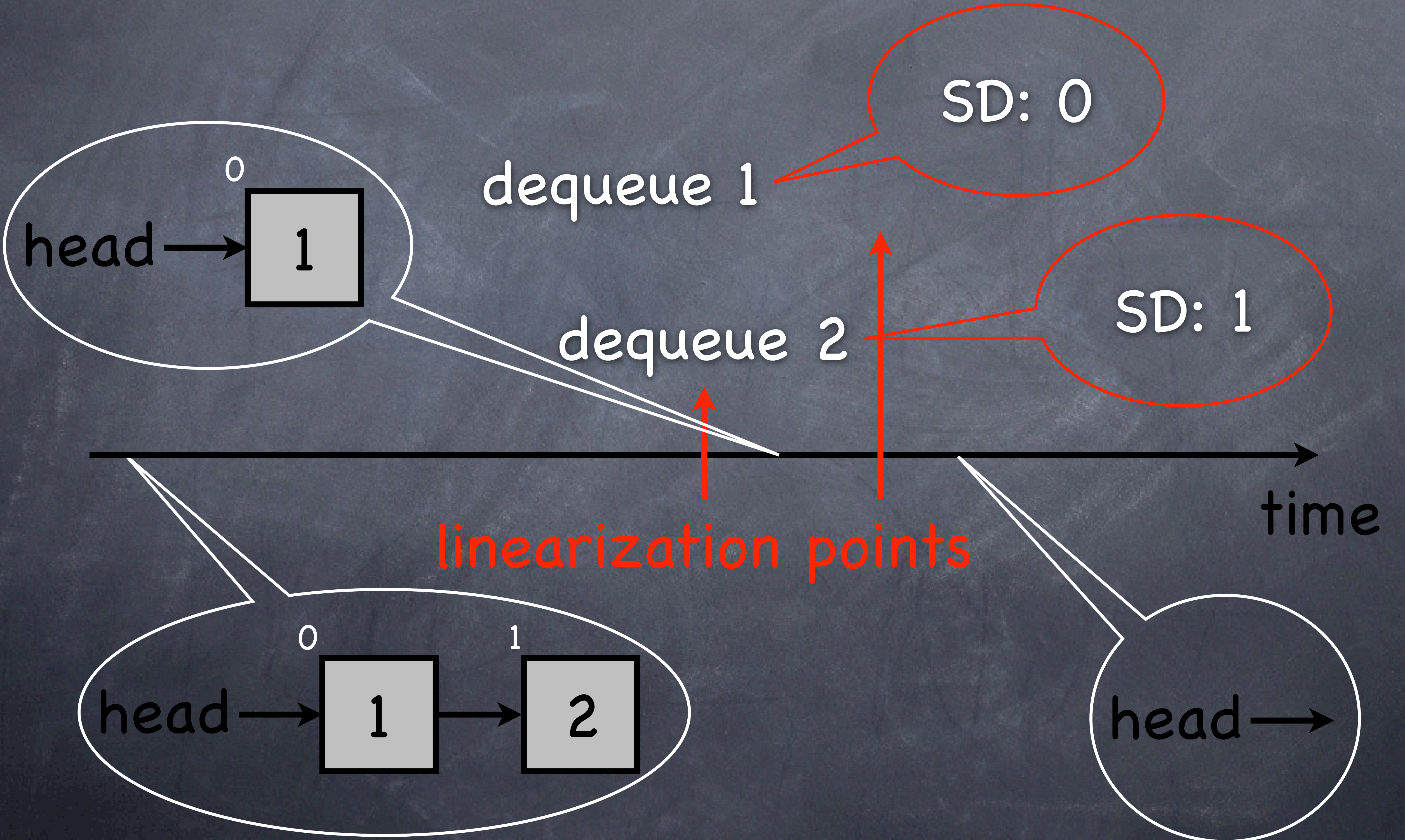
Sequential History II

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 $\leq$ ASD $\leq$ HSD

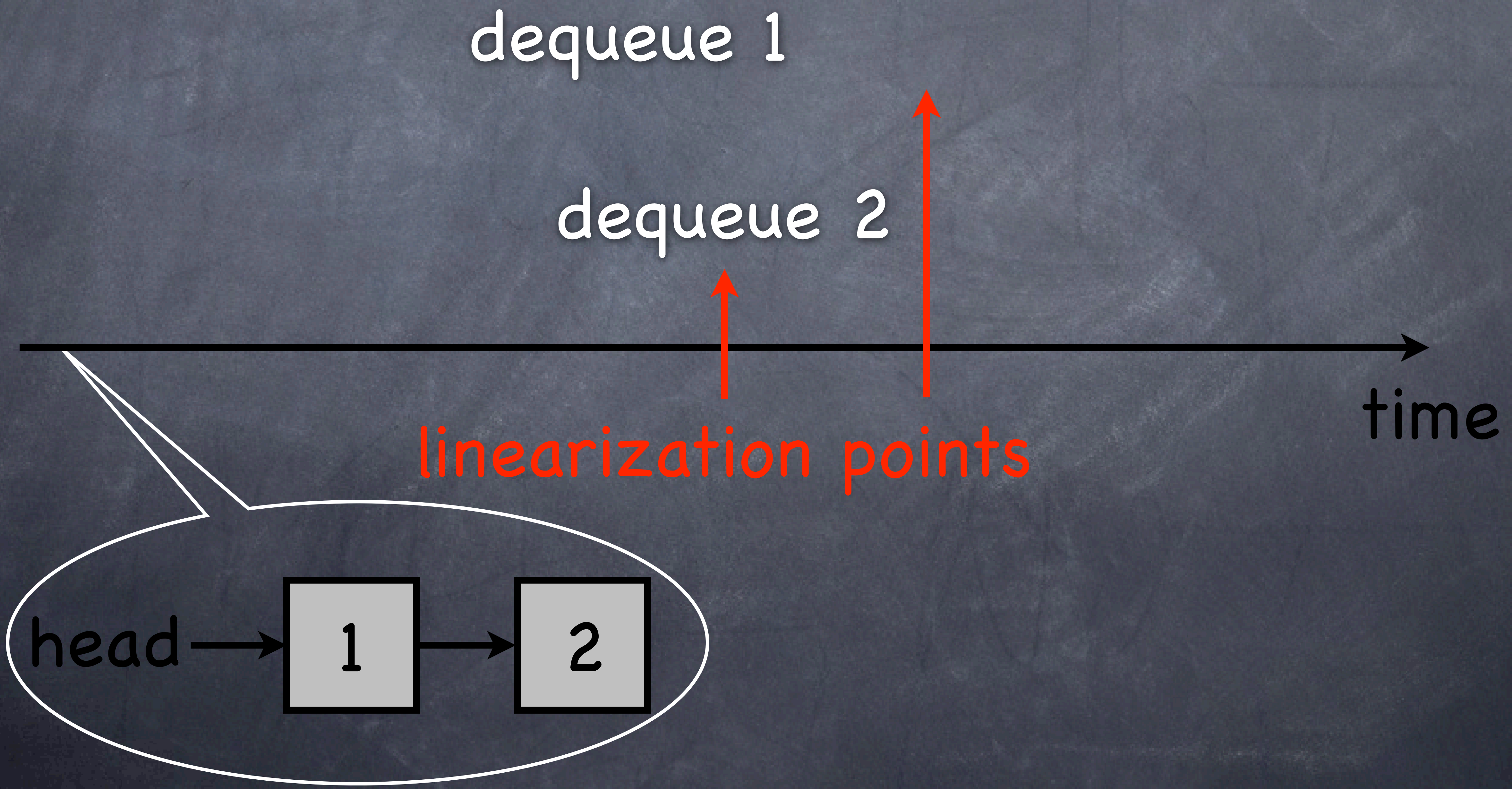
ASD Analysis

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

Proposal for large or infinite 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, response-adjusted semantical deviation (HSD) among all valid sequential histories of the execution history

ASD Analysis

Proposal for large or infinite 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, response-adjusted semantical deviation (HSD) among all valid sequential histories of the execution history

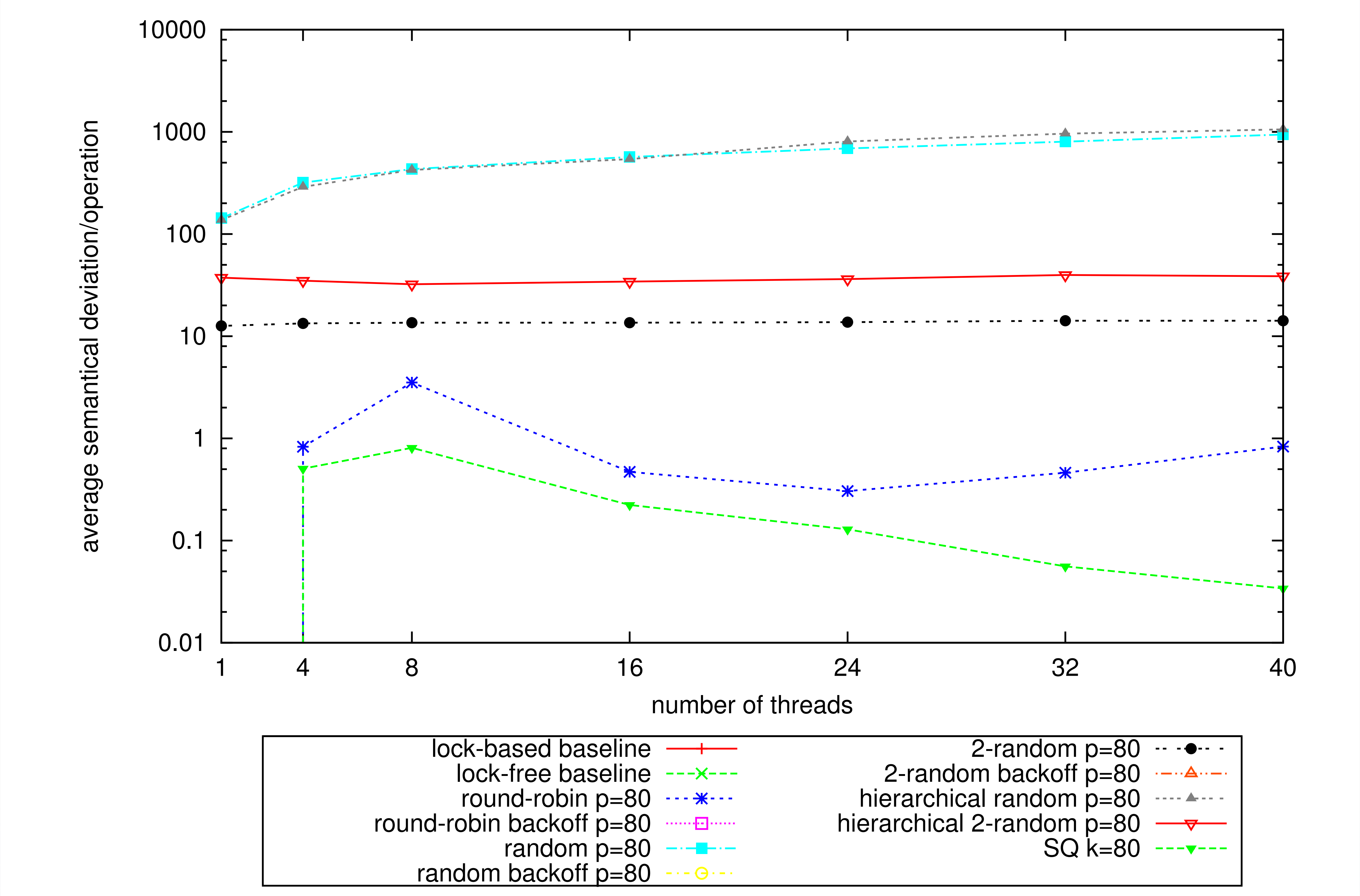
But now: ASD $\leq$ HSD may not hold

ASD Analysis

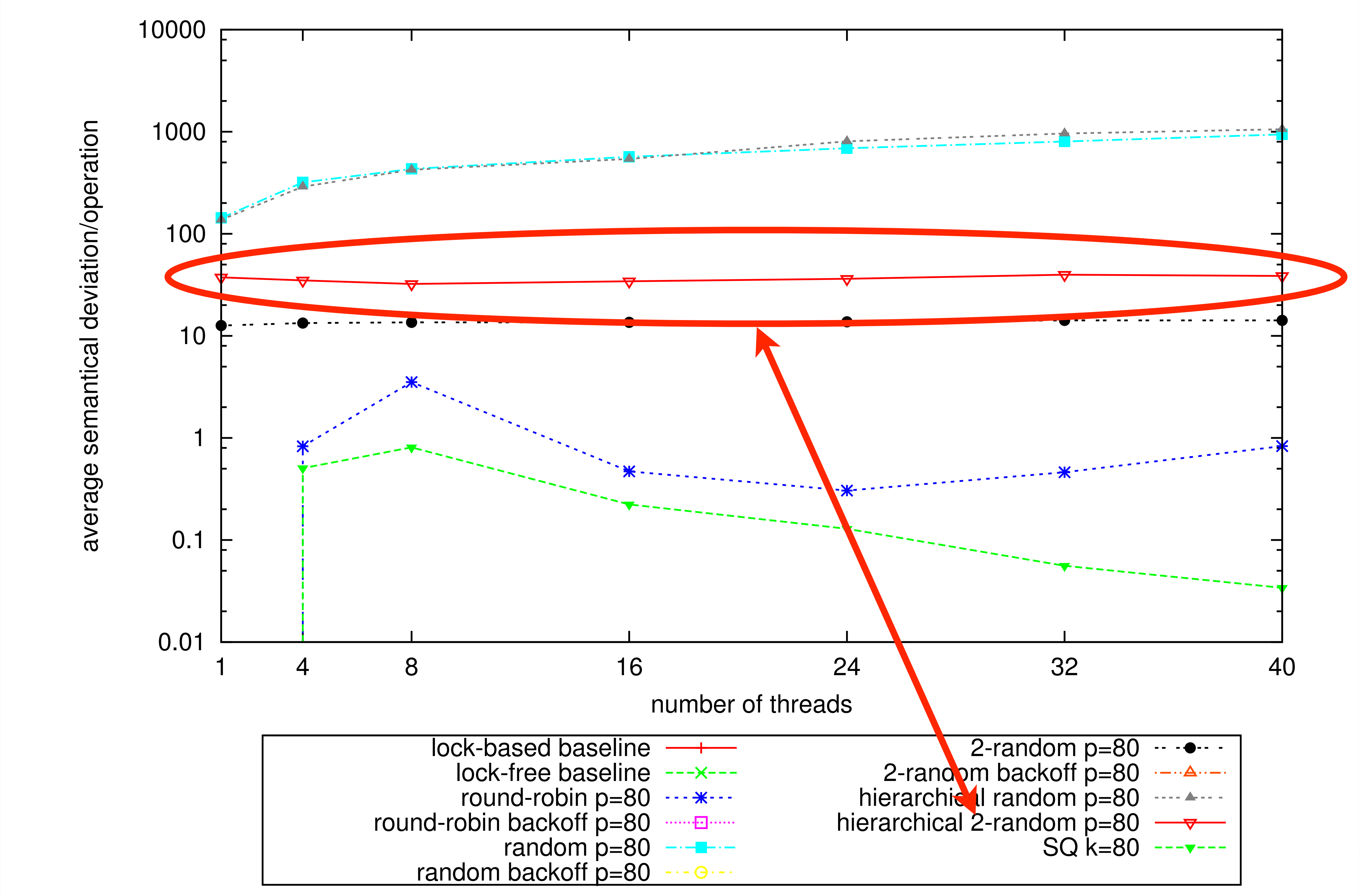
Current Version

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

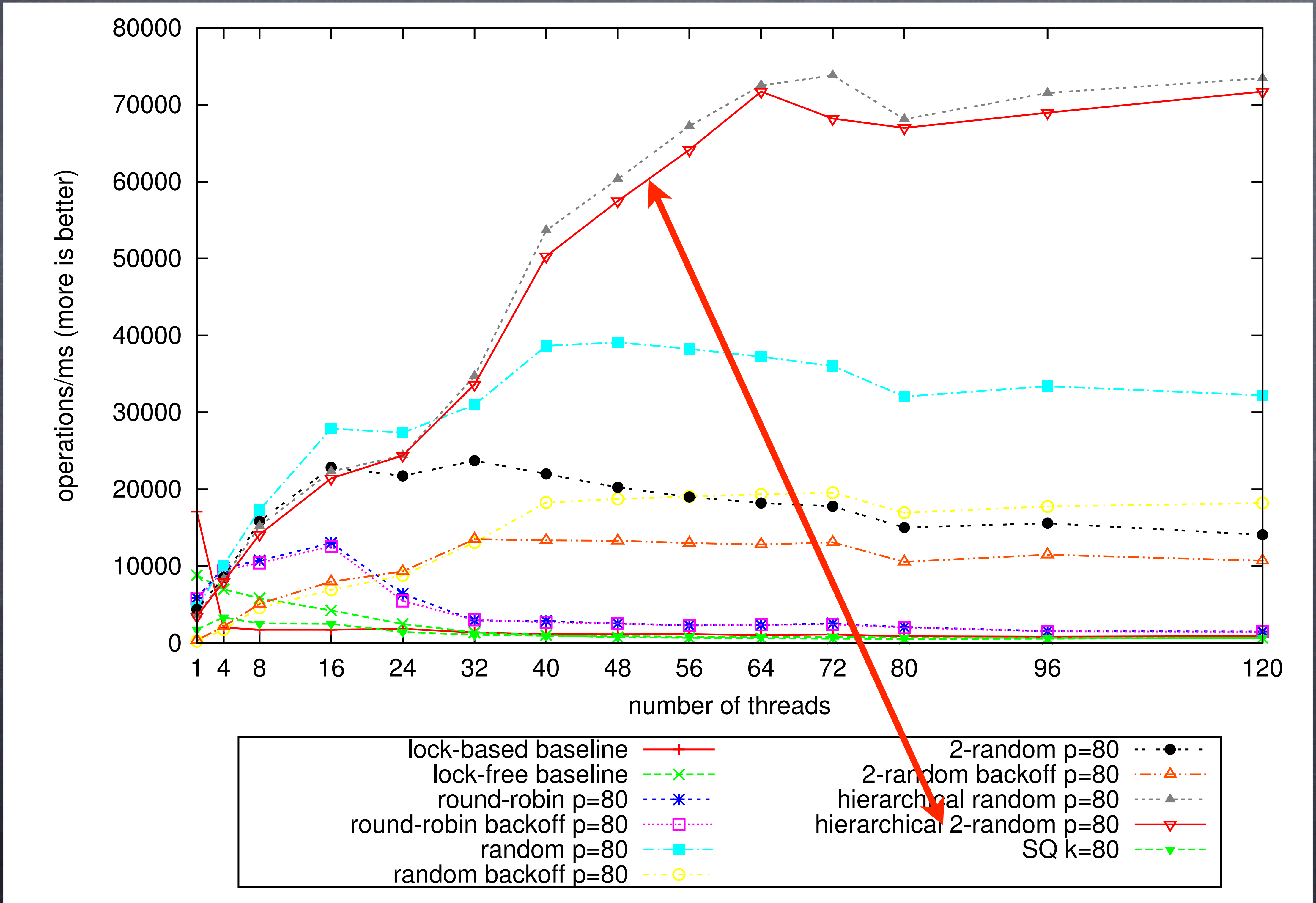
Average Semantical Deviation of LSD History



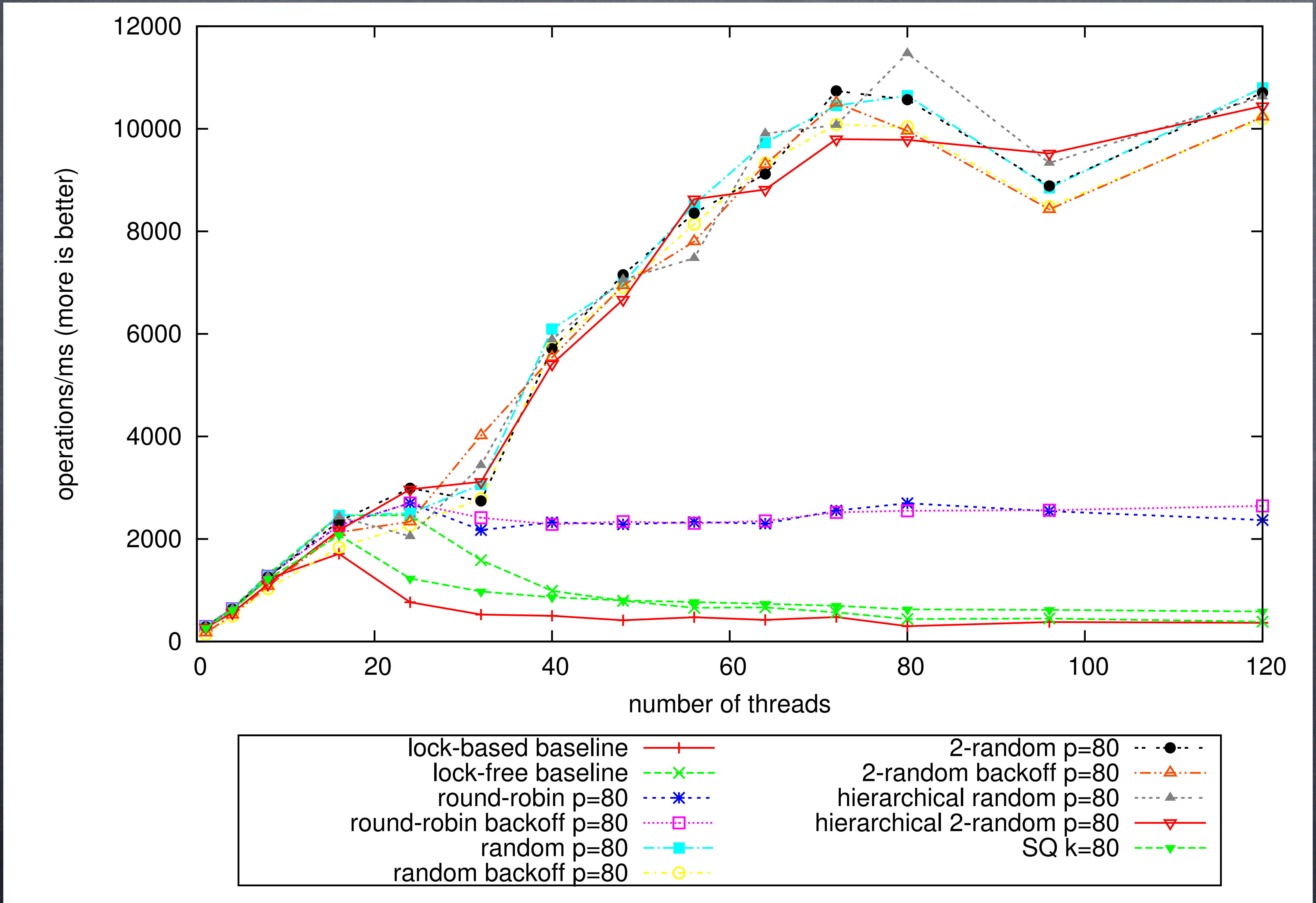
Best Trade-off



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