LITE Kernel RDMA Support for Datacenter Applications

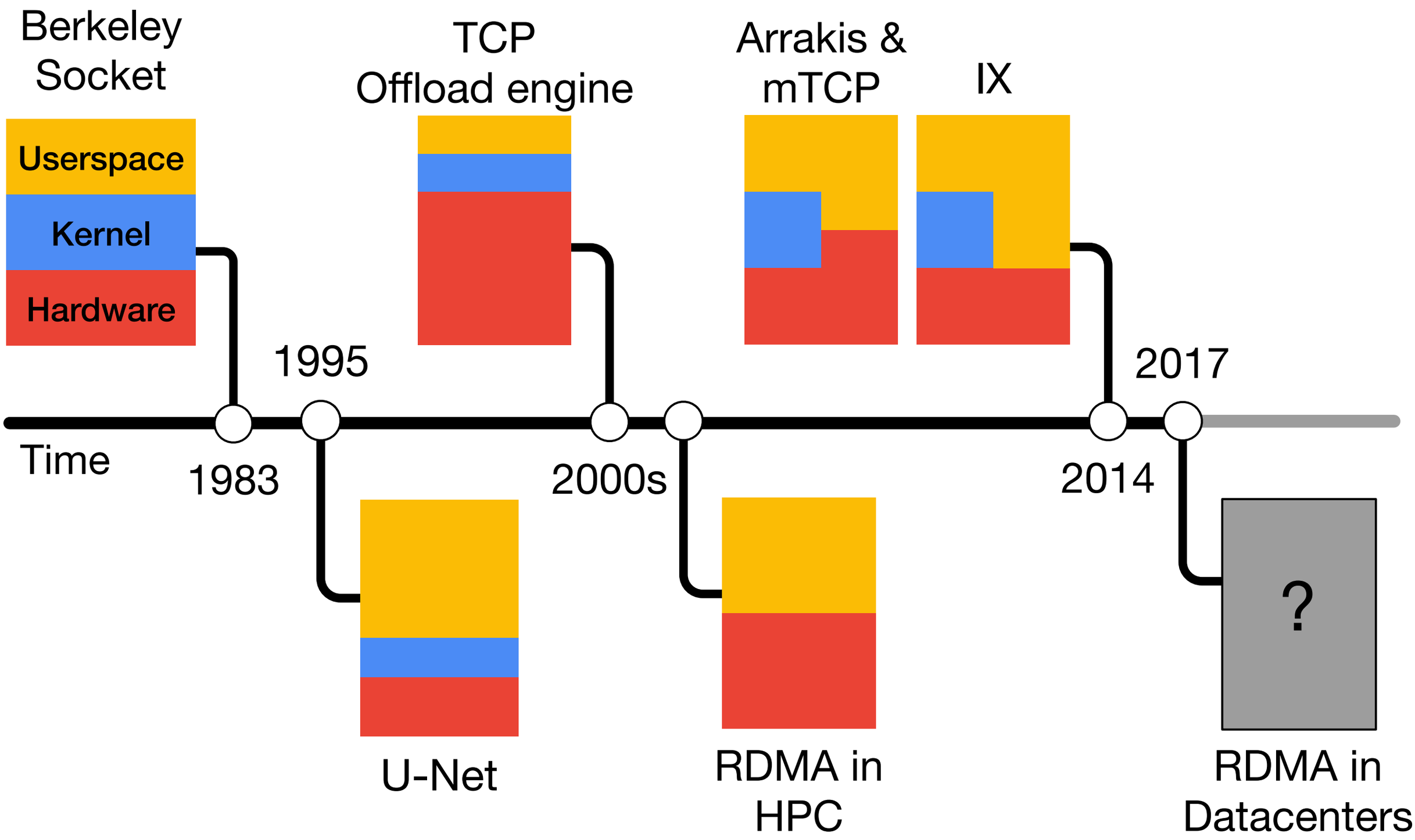
Shin-Yeh Tsai, Yiying Zhang

WukLab

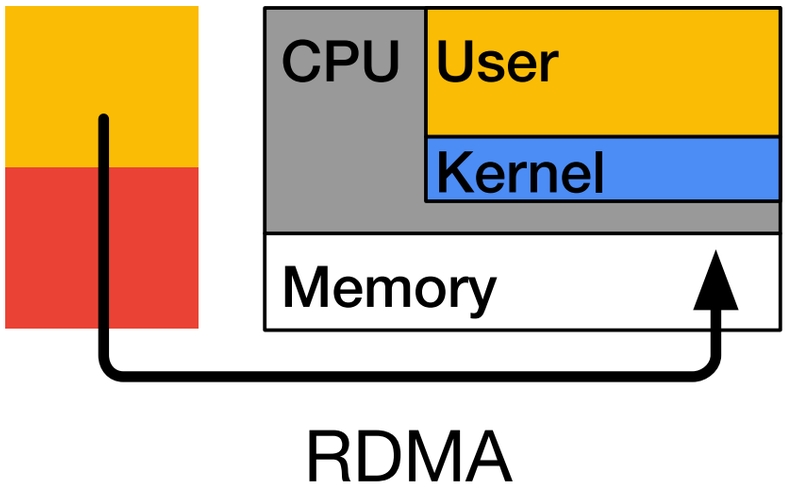
PURDUE UNIVERSITY


Time

Berkeley
Socket


Berkeley Socket
Userspace
Kernel
Hardware
TCP Offload engine
Arrakis & mTCP
IX
Time
1983
1995
2000s
2014
2017
U-Net
RDMA in HPC
?
RDMA in Datacenters

RDMA (Remote Direct Memory Access)



- Directly read/write remote memory
- Bypassing kernel
- Memory zero copy

- Benefits
  - Low latency
  - High throughput
  - Low CPU utilization

Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

RDMA-Based Datacenter Applications

RDMA-Based Datacenter Applications

Pilaf
[ATC '13]

HERD-RPC
[ATC '16]

Cell
[ATC '16]

FaRM
[NSDI '14]

Wukong
[OSDI '16]

FaSST
[OSDI '16]

HERD
[SIGCOMM '14]

Hotpot
[SoCC '17]

NAM-DB
[VLDB '17]

RSI
[VLDB '16]

DrTM
[SOSP '15]

APUS
[SoCC '17]

Octopus
[ATC '17]

DrTM+R
[EuroSys '16]

FaRM+Xact
[SOSP '15]

Mojim
[ASPLOS '15]

Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation

Native RDMA

Userspace

Hardware

User Space

Kernel Space

Hardware

User-Level RDMA App

Conn Mgmt

send

recv

node, lkey, rkey addr

Mem Mgmt

Connections

Queues

Keys

Memory space

Library

OS

RNIC

Permission check
Address mapping

Cached PTEs

lkey 1

...

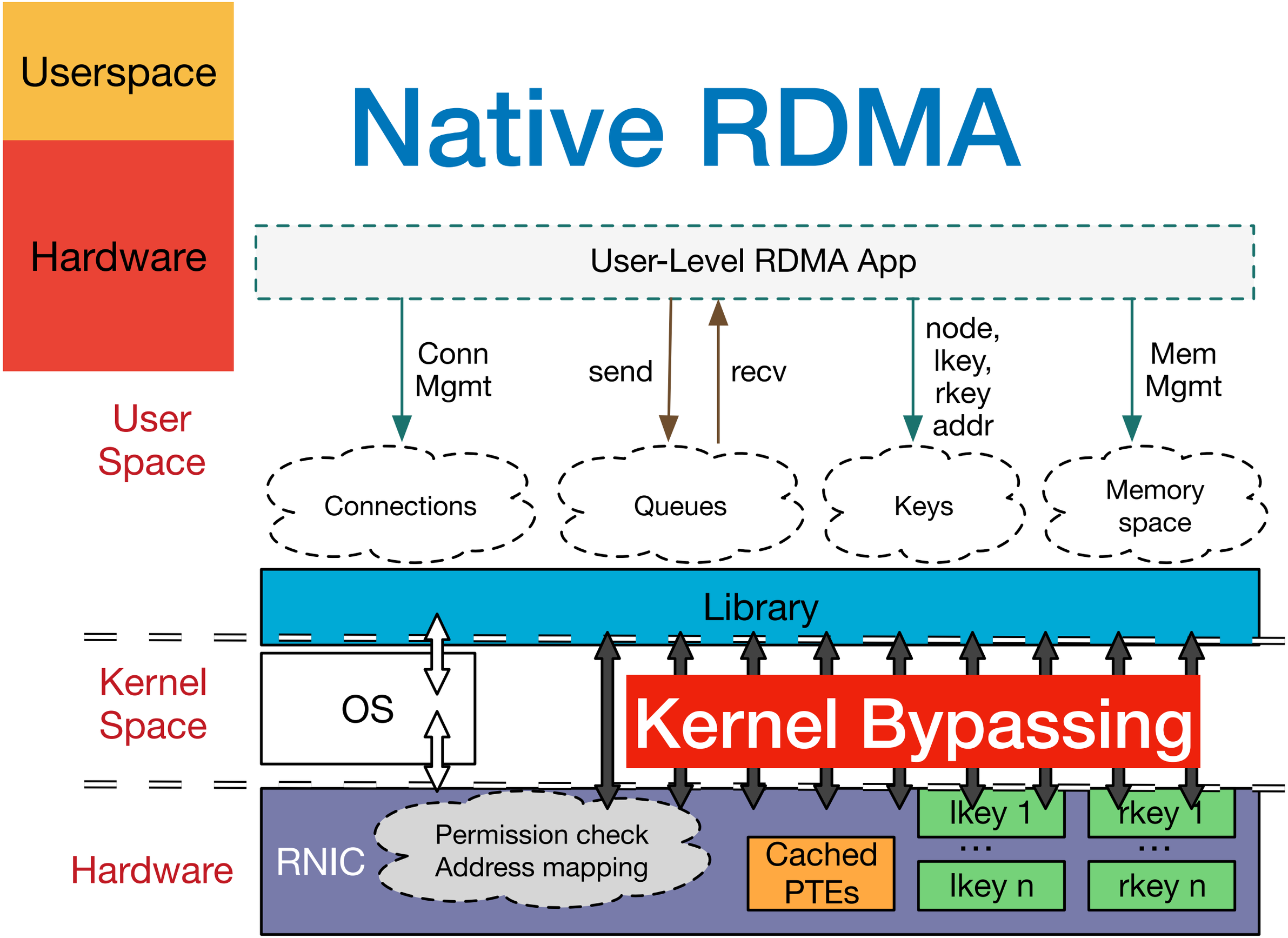
lkey n

rkey 1

...

rkey n

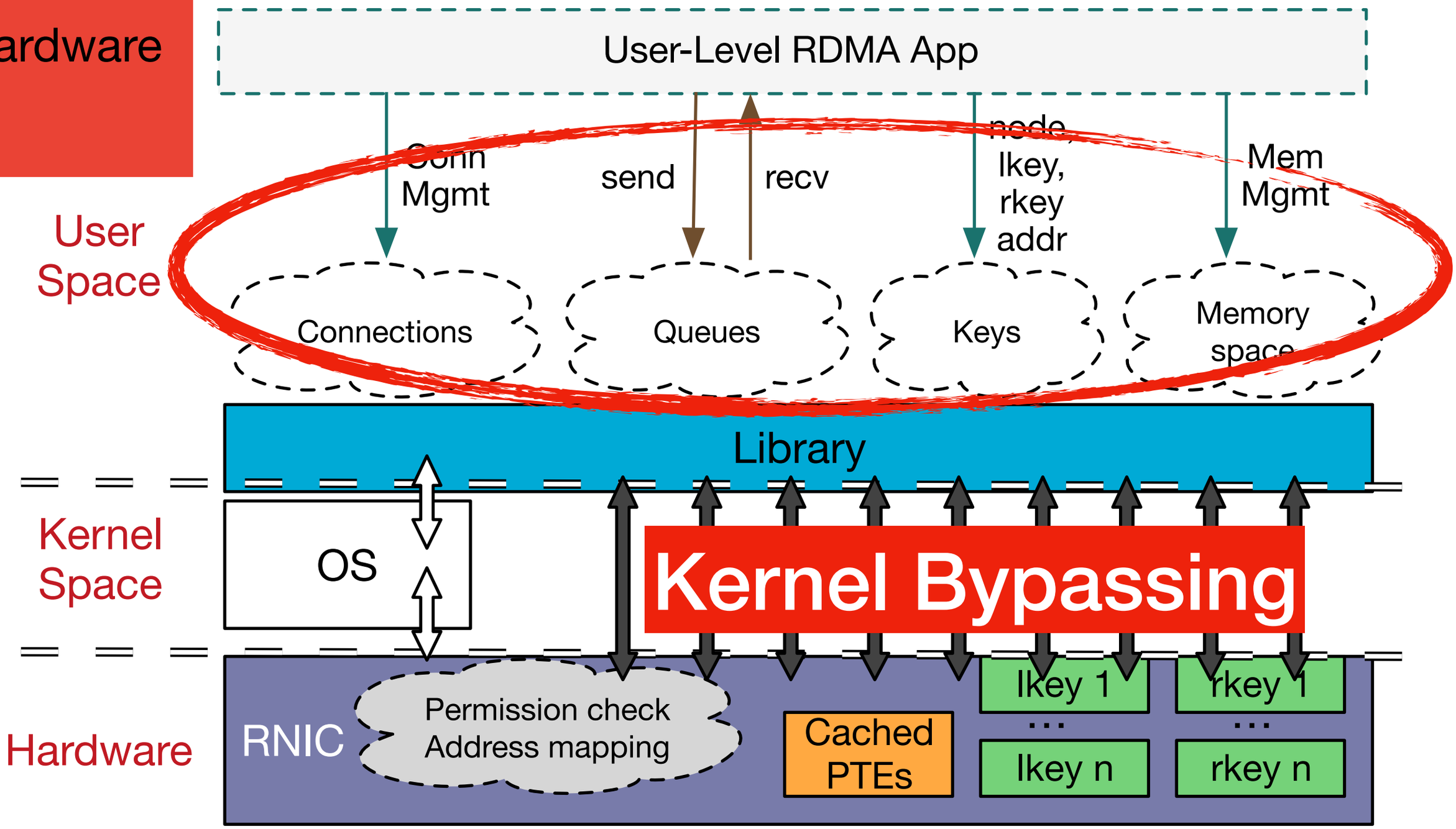
Native RDMA


Userspace
Hardware
User Space
User-Level RDMA App
Conn Mgmt
send
recv
node, lkey, rkey addr
Mem Mgmt
Connections
Queues
Keys
Memory space
Library
Kernel Space
OS
Kernel Bypassing
Hardware
RNIC
Permission check
Address mapping
Cached PTEs
lkey 1
lkey n
rkey 1
rkey n

Userspace

Hardware

Native RDMA



Userspace

Hardware

High-level

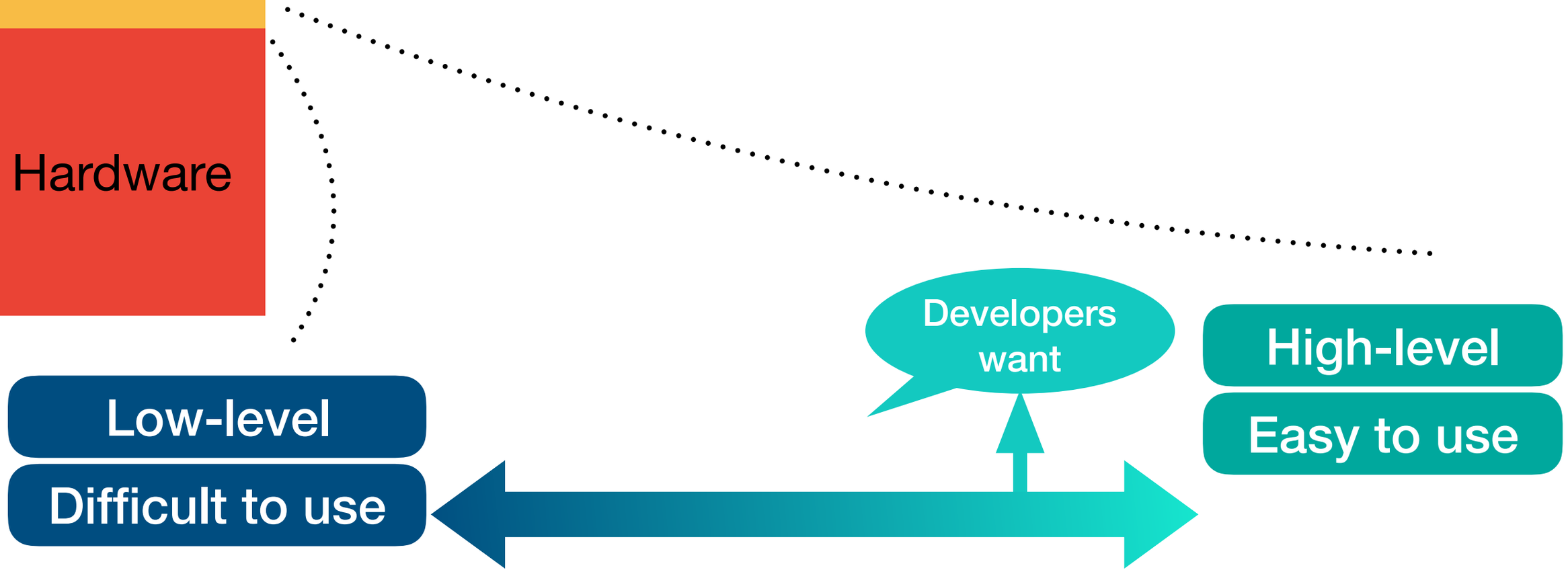
Easy to use

Low-level

Difficult to use

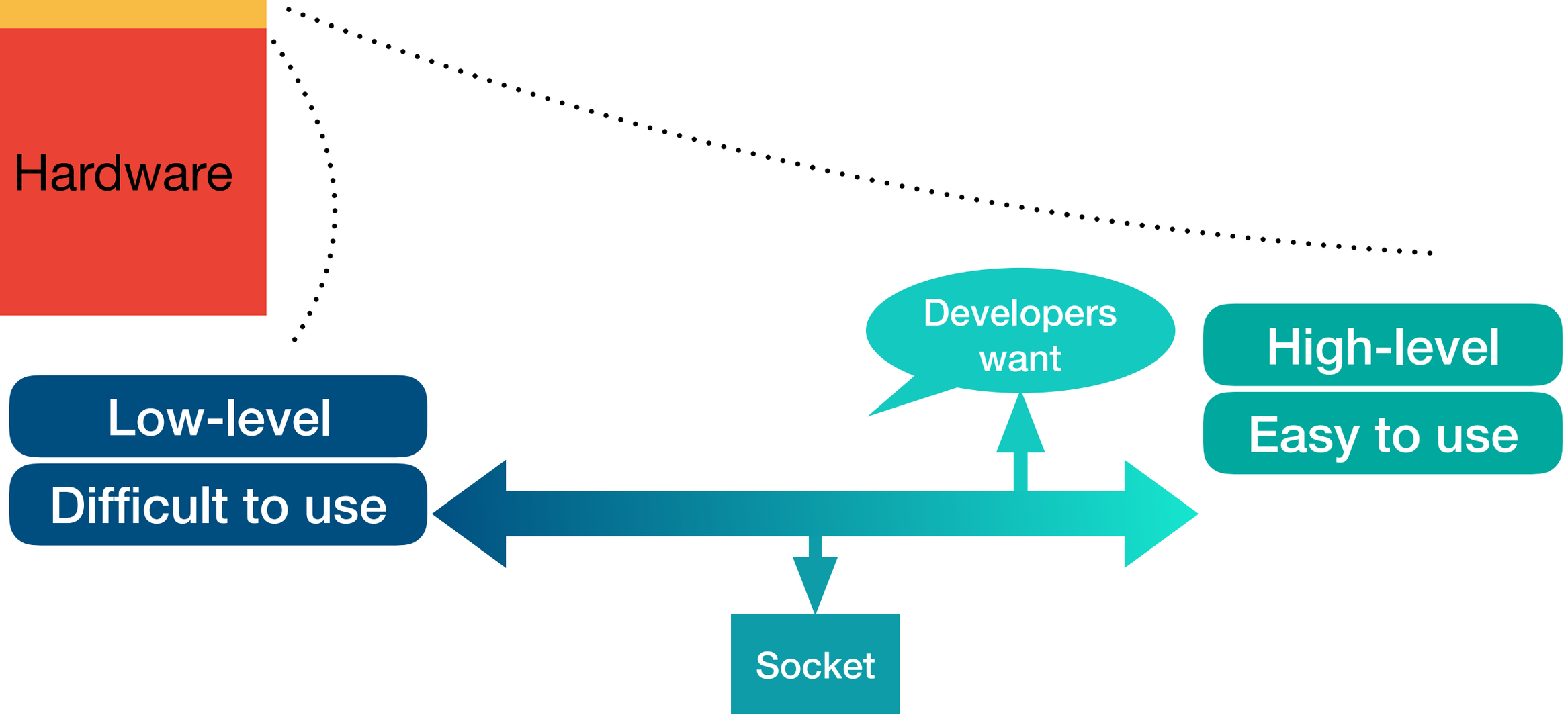
Userspace

Hardware



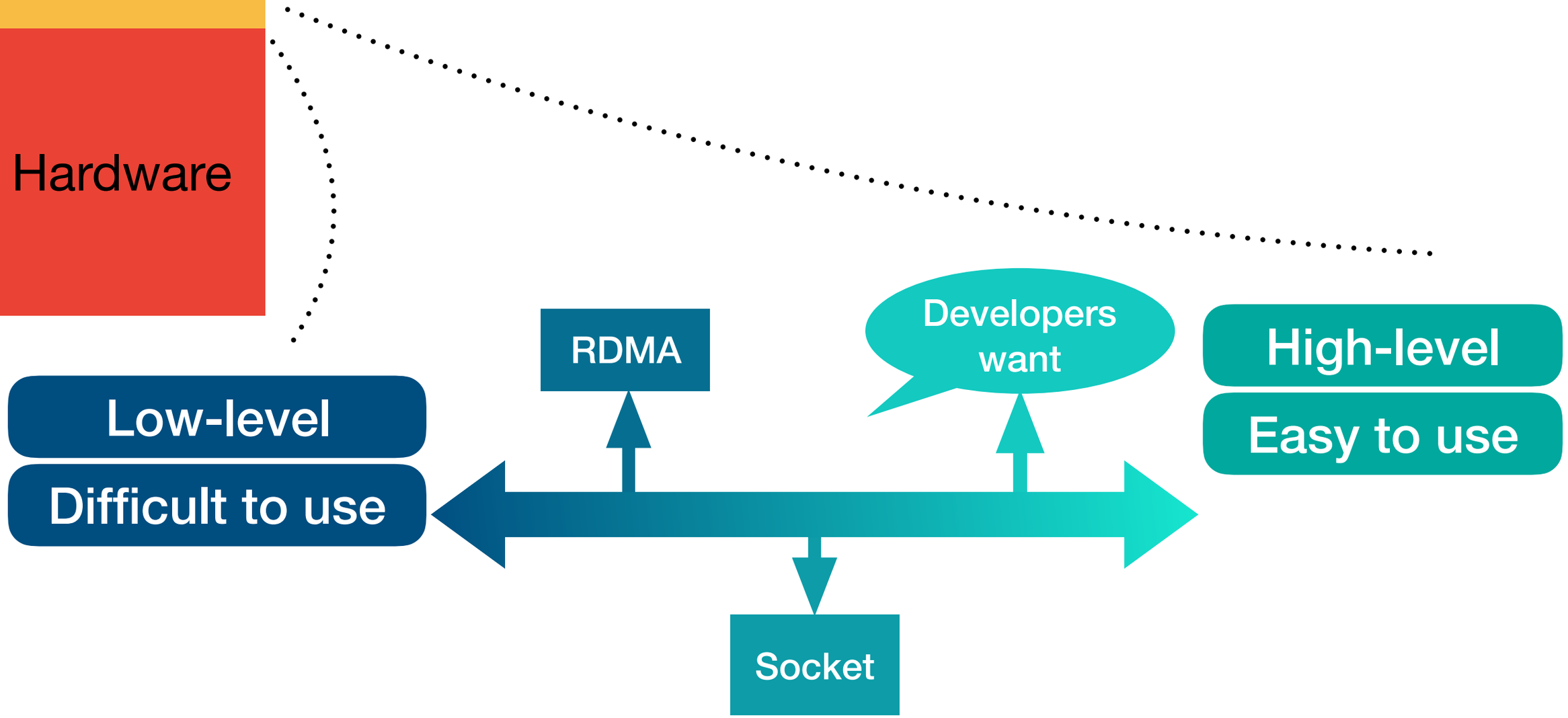
Userspace

Hardware



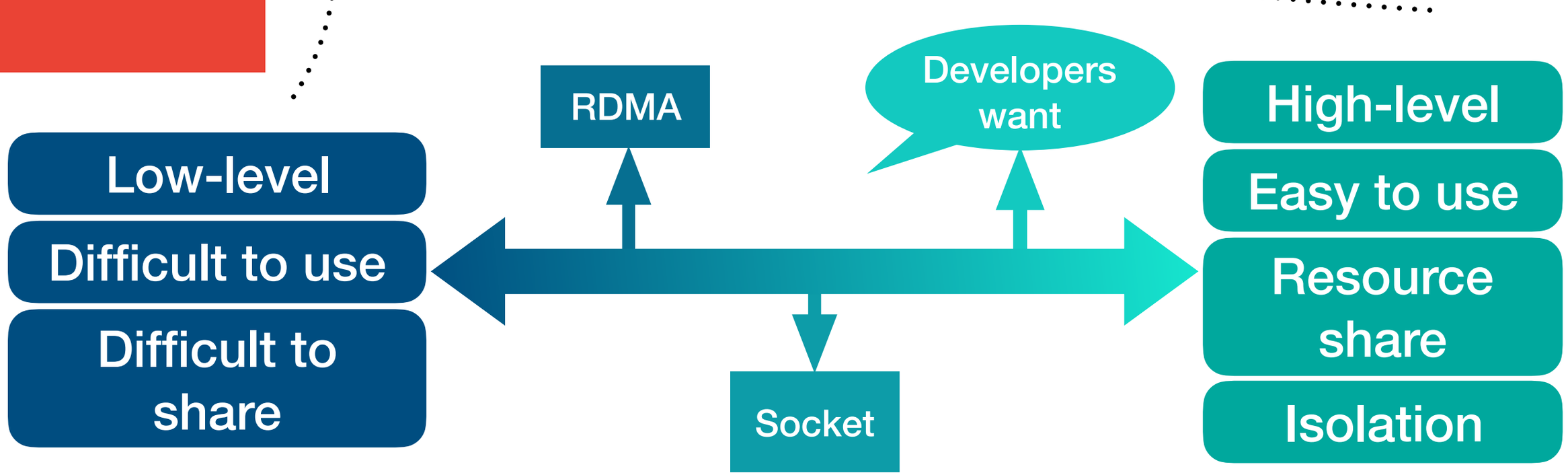
Userspace

Hardware



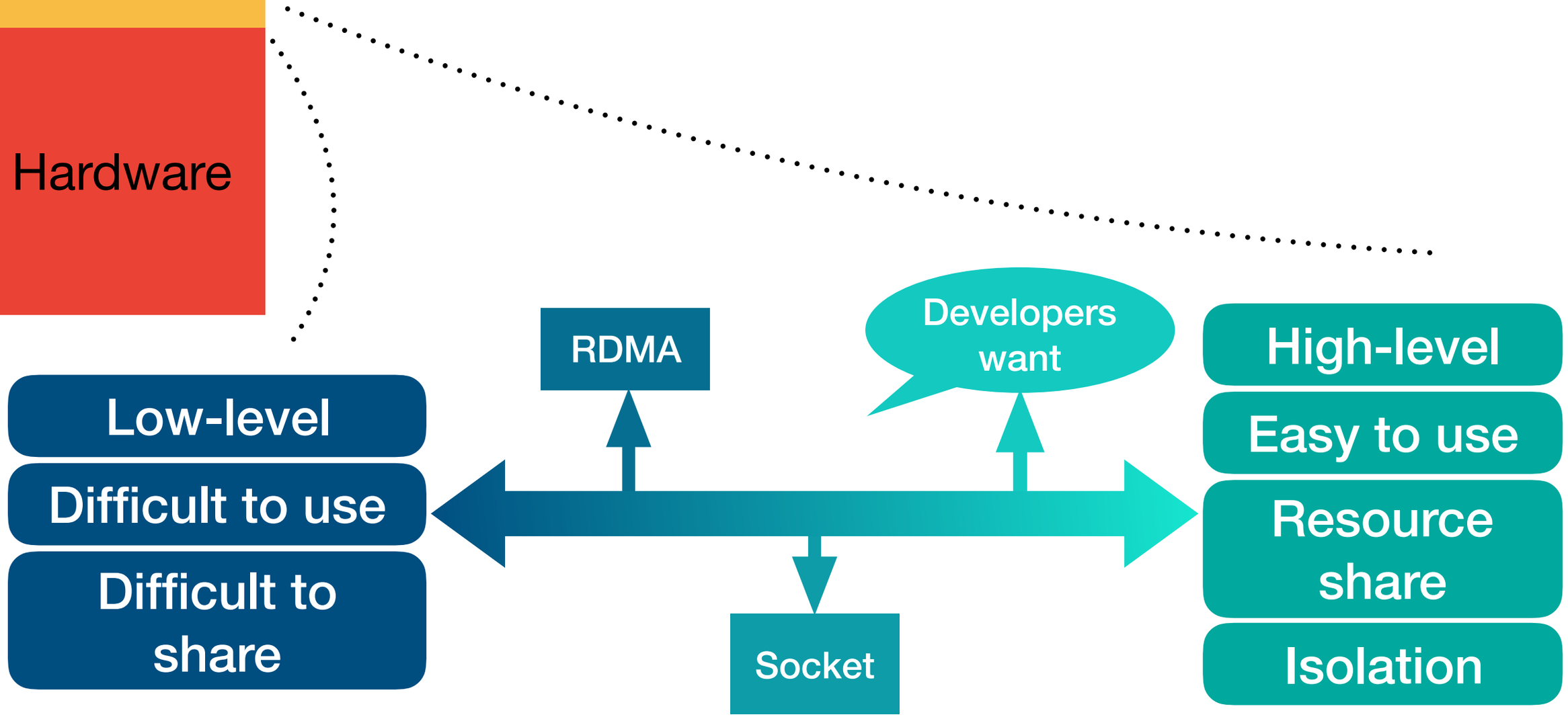
Userspace

Hardware



Userspace

Hardware



Abstraction Mismatch

Userspace

Hardware

Fat applications
No resource sharing

Low-level

Difficult to use

Difficult to share

RDMA

Socket

Developers want

High-level

Easy to use

Resource share

Isolation

Abstraction Mismatch

Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation

Things have worked well in HPC

- Special hardware
- Few applications 
- Cheaper developer 

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation

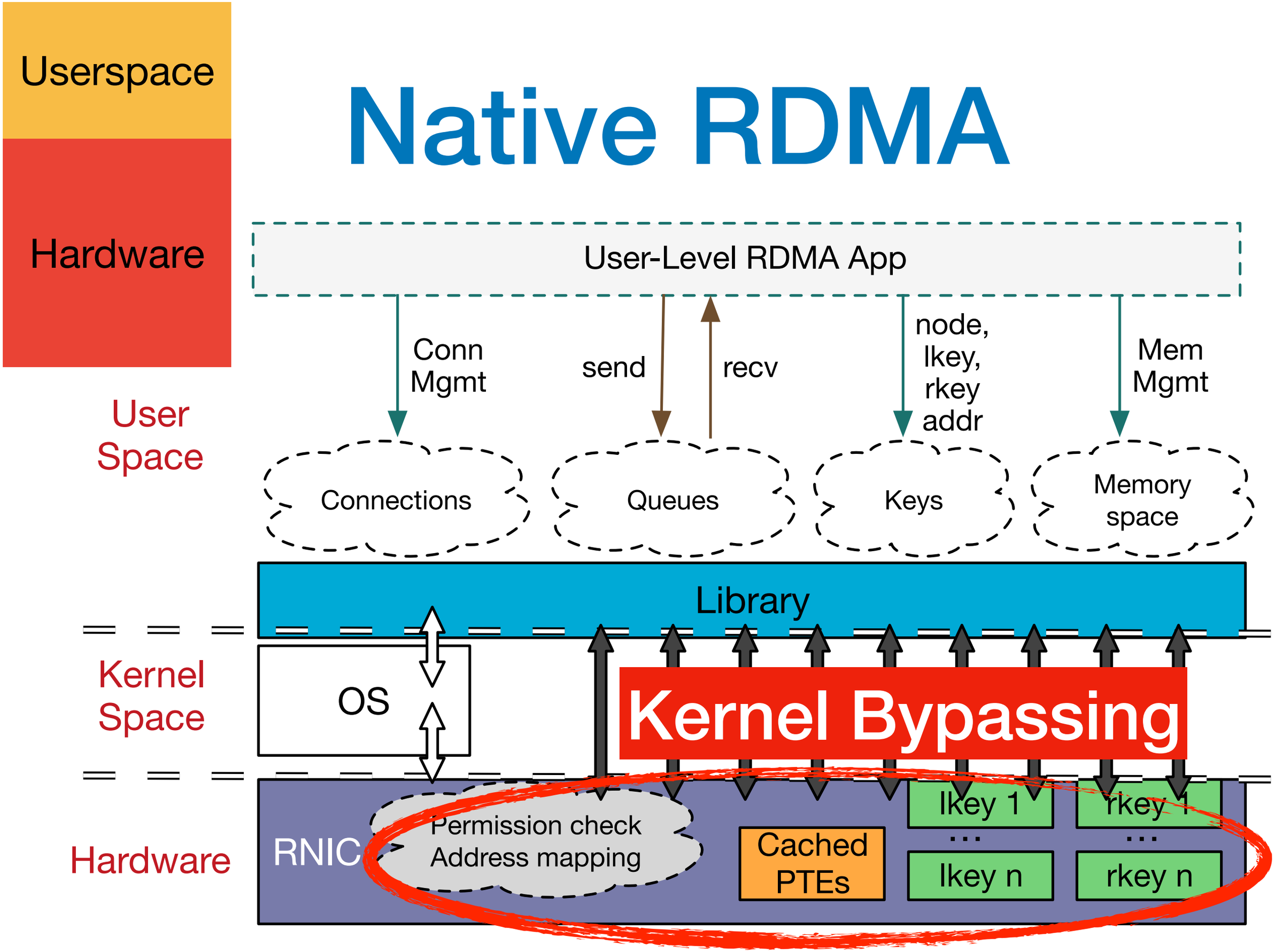
Things have worked well in HPC

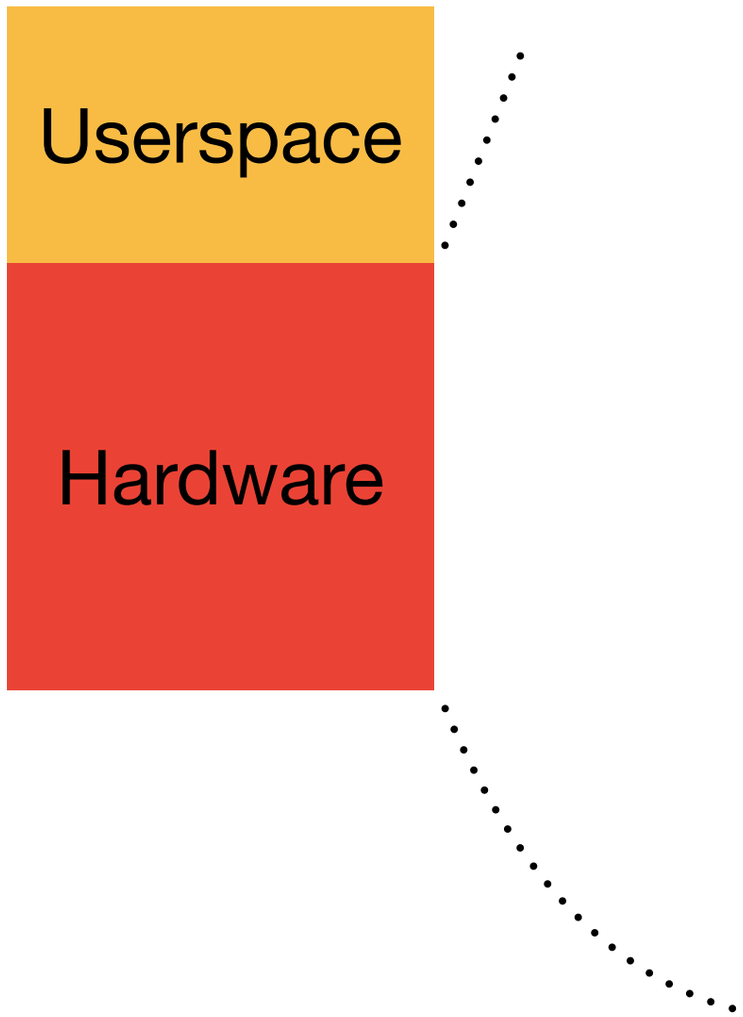
- Special hardware
- Few applications 
- Cheaper developer 

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications 
- Resource sharing and isolation 

Native RDMA


Userspace
Hardware
User Space
User-Level RDMA App
Conn Mgmt
send
recv
node, lkey, rkey addr
Mem Mgmt
Connections
Queues
Keys
Memory space
Library
Kernel Space
OS
Kernel Bypassing
Hardware
RNIC
Permission check
Address mapping
Cached PTEs
lkey 1
lkey n
rkey 1
rkey n


Userspace
Hardware

On-NIC SRAM

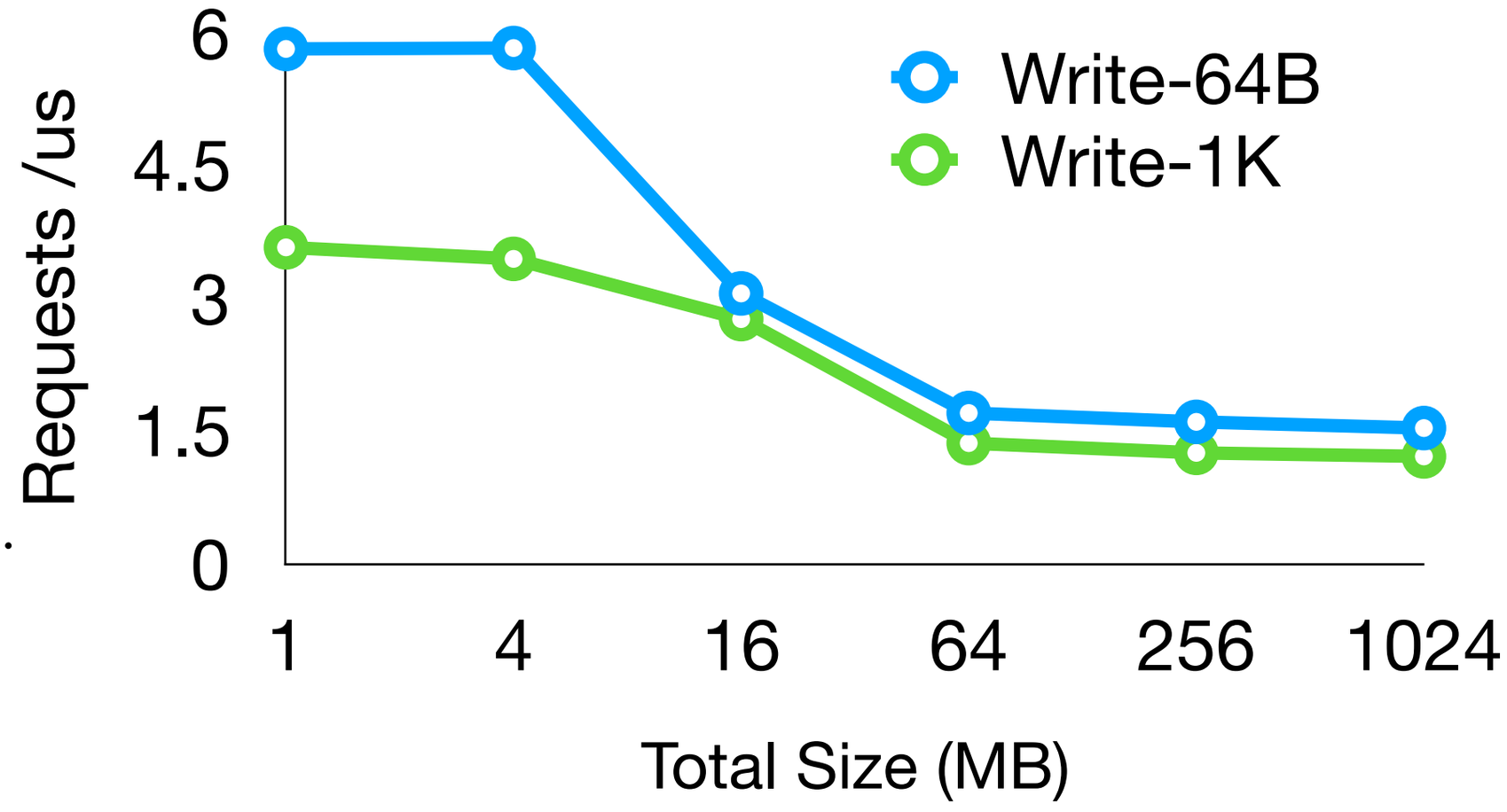
1. Fetches and caches page table entries
2. Stores secret keys for **every** consecutive **memory region**

Userspace

Hardware

On-NIC SRAM

1. Fetches and caches page table entries
2. Stores secret keys for **every** consecutive **memory region**



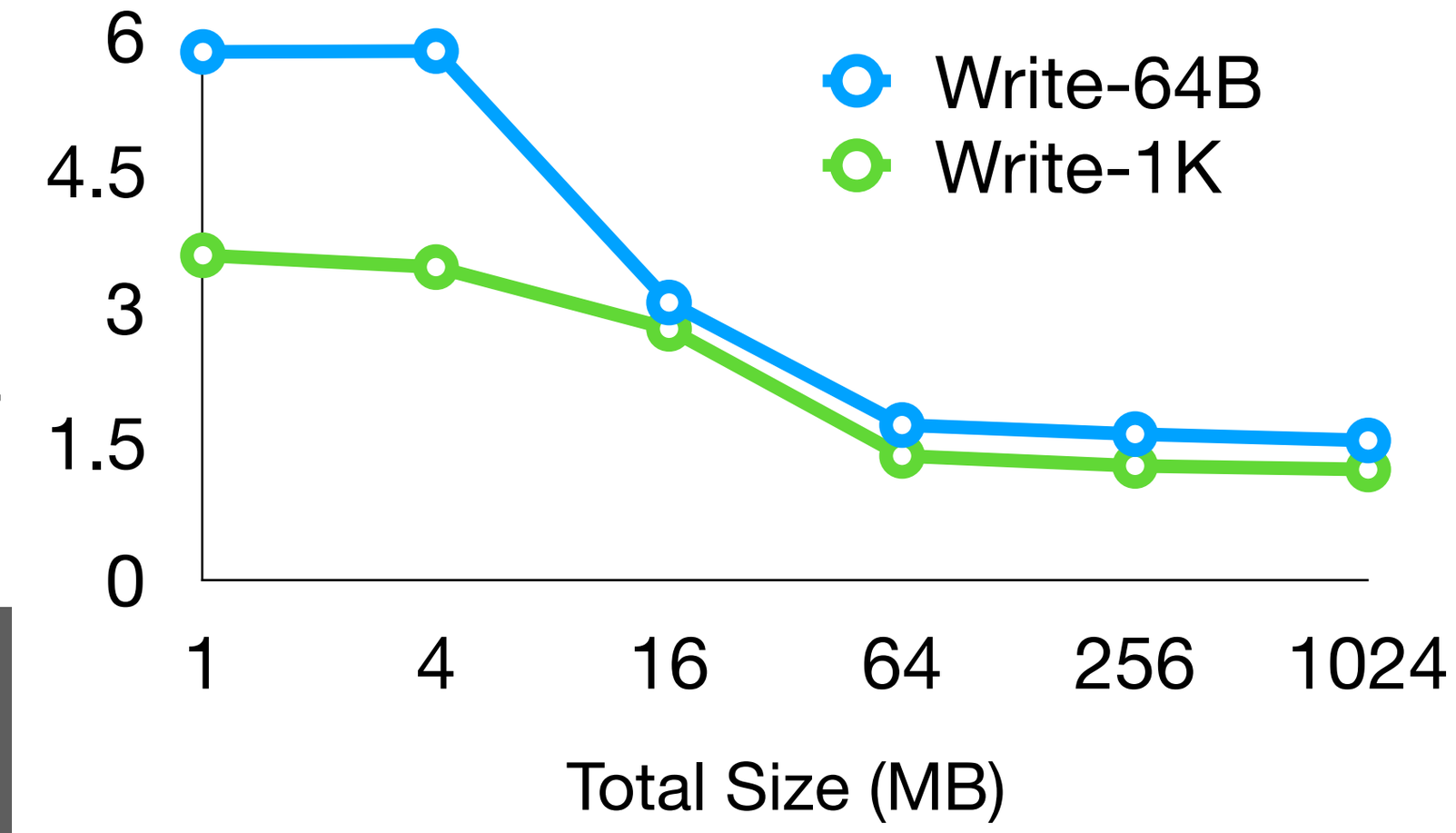
Userspace

Hardware

Expensive, unscalable hardware

On-NIC SRAM

1. Fetches and caches page table entries
2. Stores secret keys for **every** consecutive **memory region**



Things have been good in HPC

- Special hardware
- Few applications 
- Cheaper developer 

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications 
- Resource sharing and isolation 

Things have been good in HPC

- Special hardware 
- Few applications 
- Cheaper developer 

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications 
- Resource sharing and isolation 

Things have been good in HPC

- Special hardware 
- Few applications 
- Cheaper developer 

What about datacenters?

- Commodity, cheaper hardware 
- Many (changing) applications 
- Resource sharing and isolation 

Fat applications No resource sharing

Expensive, unscalable hardware

Fat applications No resource sharing

Expensive, unscalable hardware

Are we removing too much from kernel?

Outline

- Introduction and motivation
- **Overall design and abstraction**
- **LITE internals**
- **LITE applications**
- **Conclusion**

Without Kernel

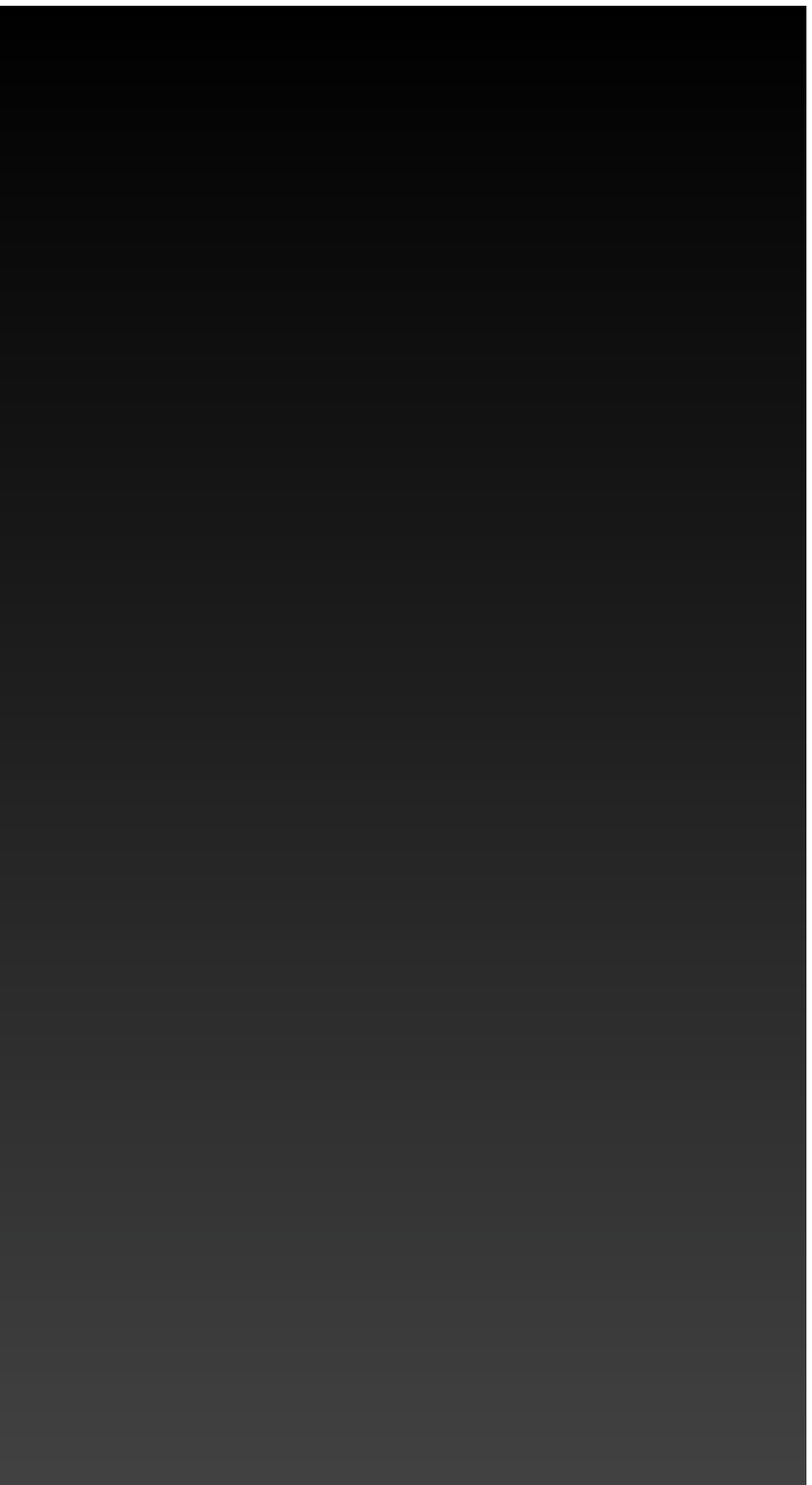
High-level abstraction

Resource sharing

Protection

Performance isolation

Without Kernel

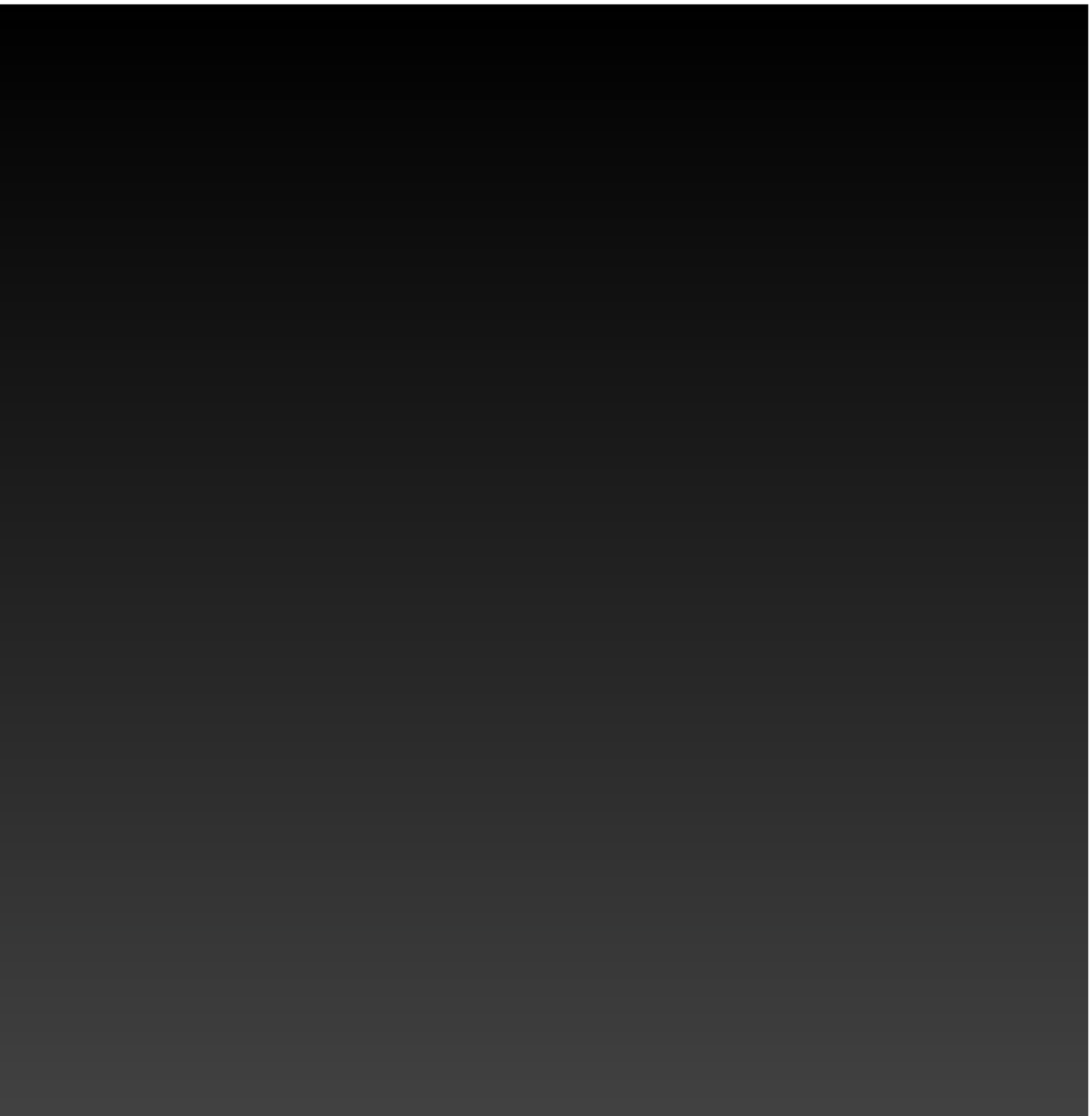


Resource sharing

Protection

Performance isolation

Without Kernel



Resource sharing

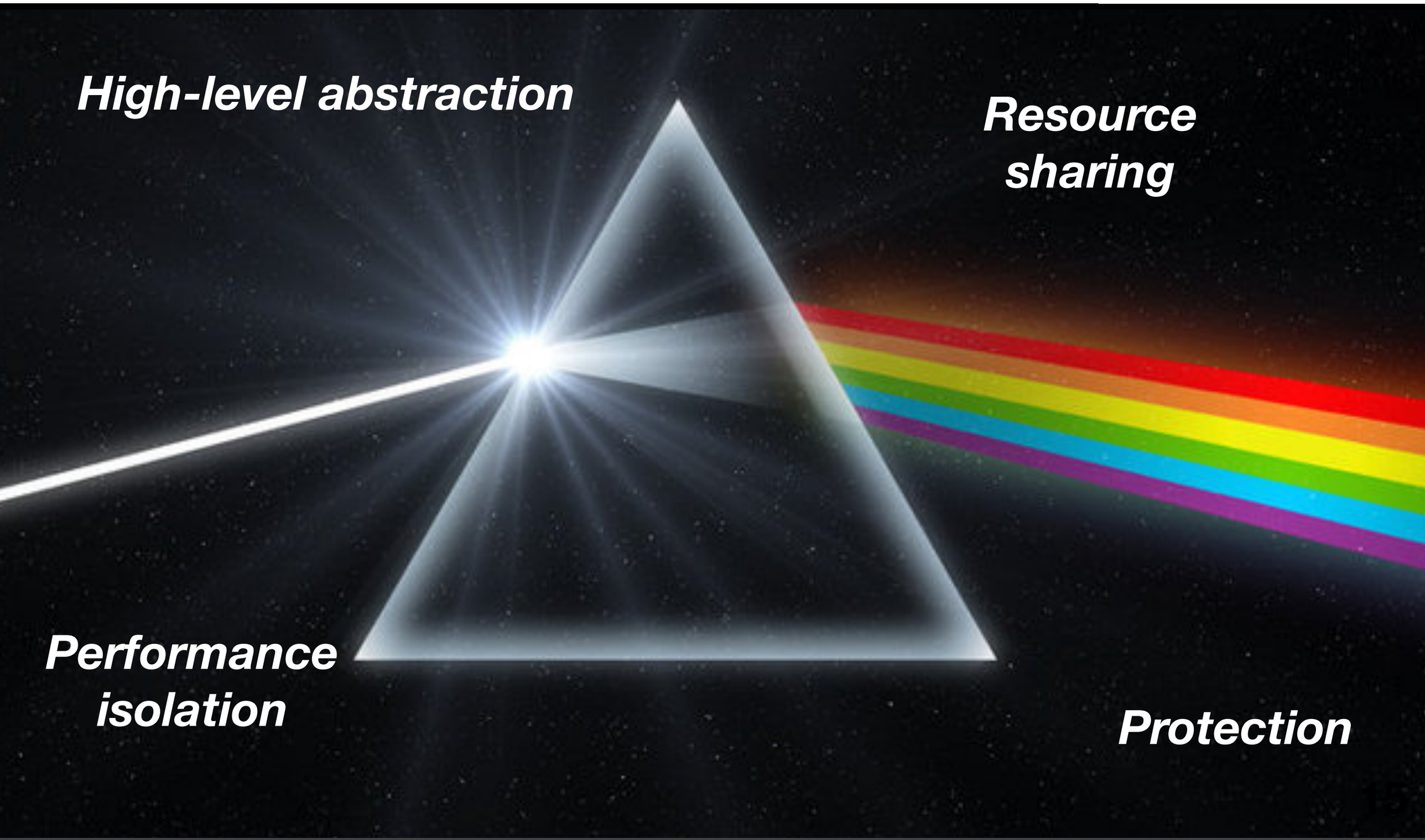
Performance isolation

Without Kernel

Performance isolation

Without Kernel

LITE - Local Indirection TiEr



All problems in computer science can be solved by another level of indirection

Butler Lampson

User-Level RDMA App

Conn Mgmt

send

recv

node, lkey, rkey addr

Mem Mgmt

User Space

Connections

Queues

Keys

Memory space

Library

Hardware

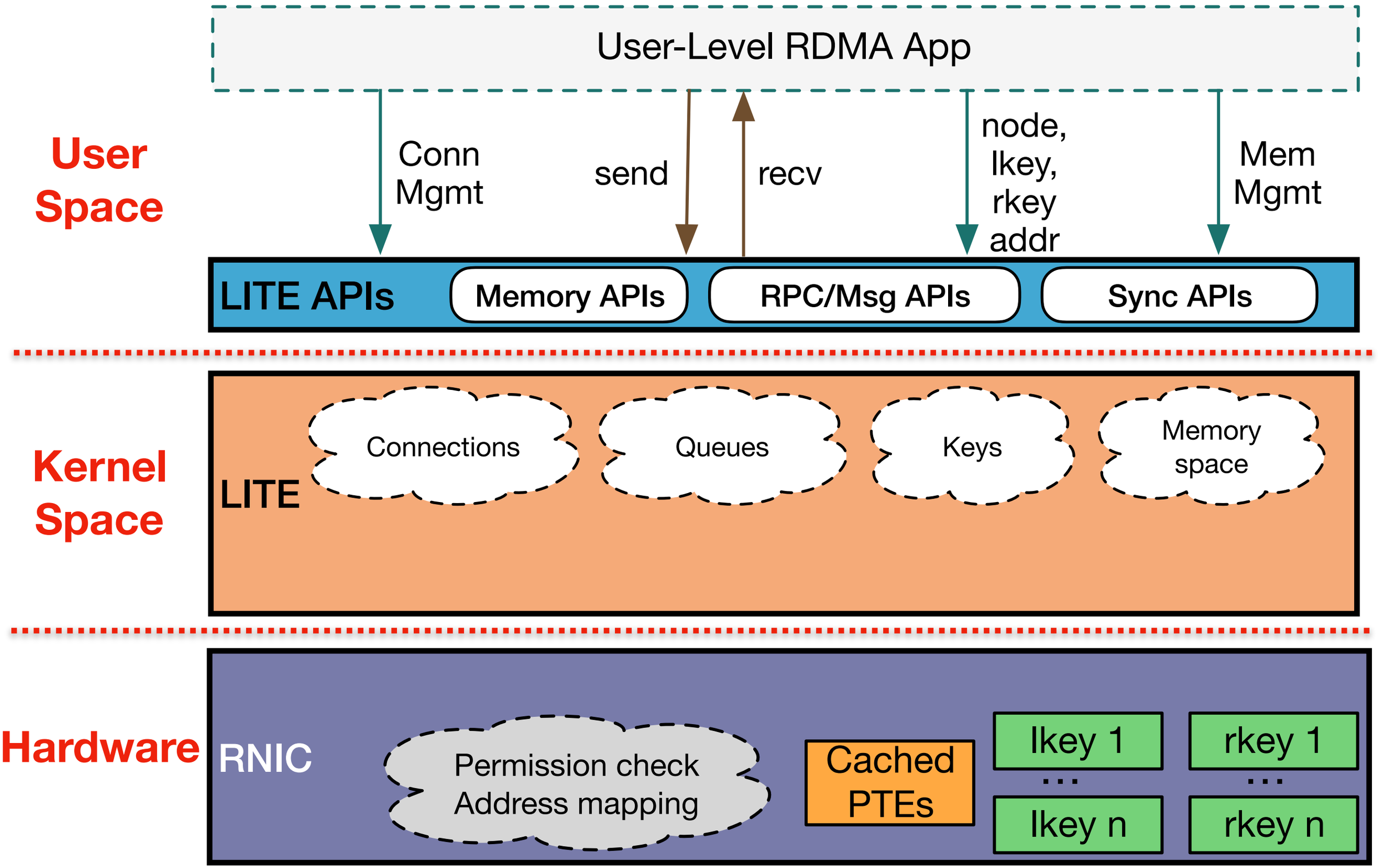
RNIC

Permission check
Address mapping

Cached PTEs

lkey 1
...
lkey n

rkey 1
...
rkey n


User Space
User-Level RDMA App
Conn Mgmt
send
recv
node, lkey, rkey addr
Mem Mgmt
LITE APIs
Memory APIs
RPC/Msg APIs
Sync APIs
Kernel Space
LITE
Connections
Queues
Keys
Memory space
Hardware
RNIC
Permission check
Address mapping
Cached PTEs
lkey 1
lkey n
rkey 1
rkey n

Simpler applications

User Space

User-Level RDMA App

Conn Mgmt

send

recv

node, lkey, rkey addr

Mem Mgmt

LITE APIs

Memory APIs

RPC/Msg APIs

Sync APIs

Kernel Space

LITE

Connections

Queues

Keys

Memory space

Hardware

RNIC

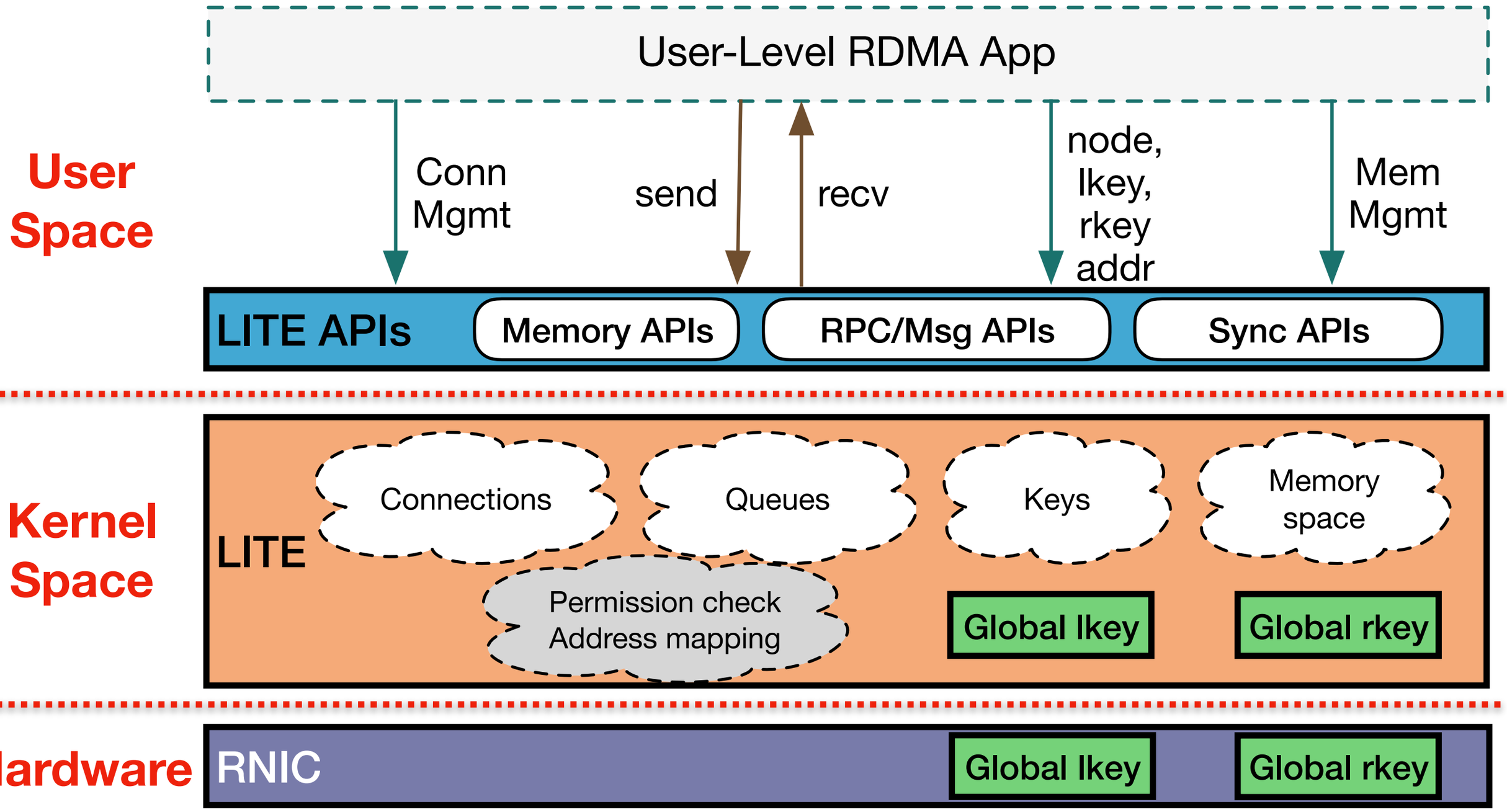
Permission check
Address mapping

Cached PTEs

lkey 1
…
lkey n

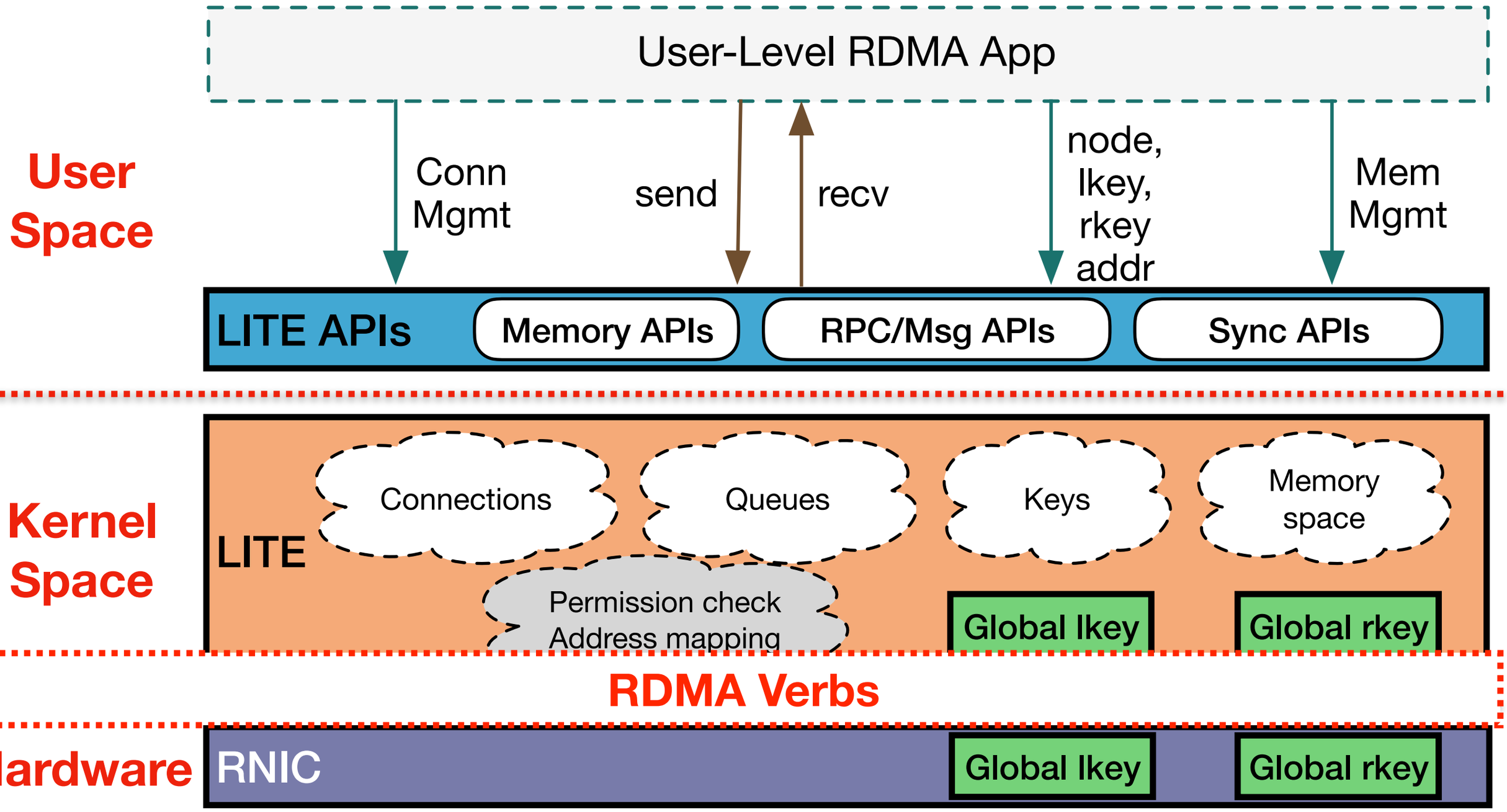
rkey 1
…
rkey n

Simpler applications



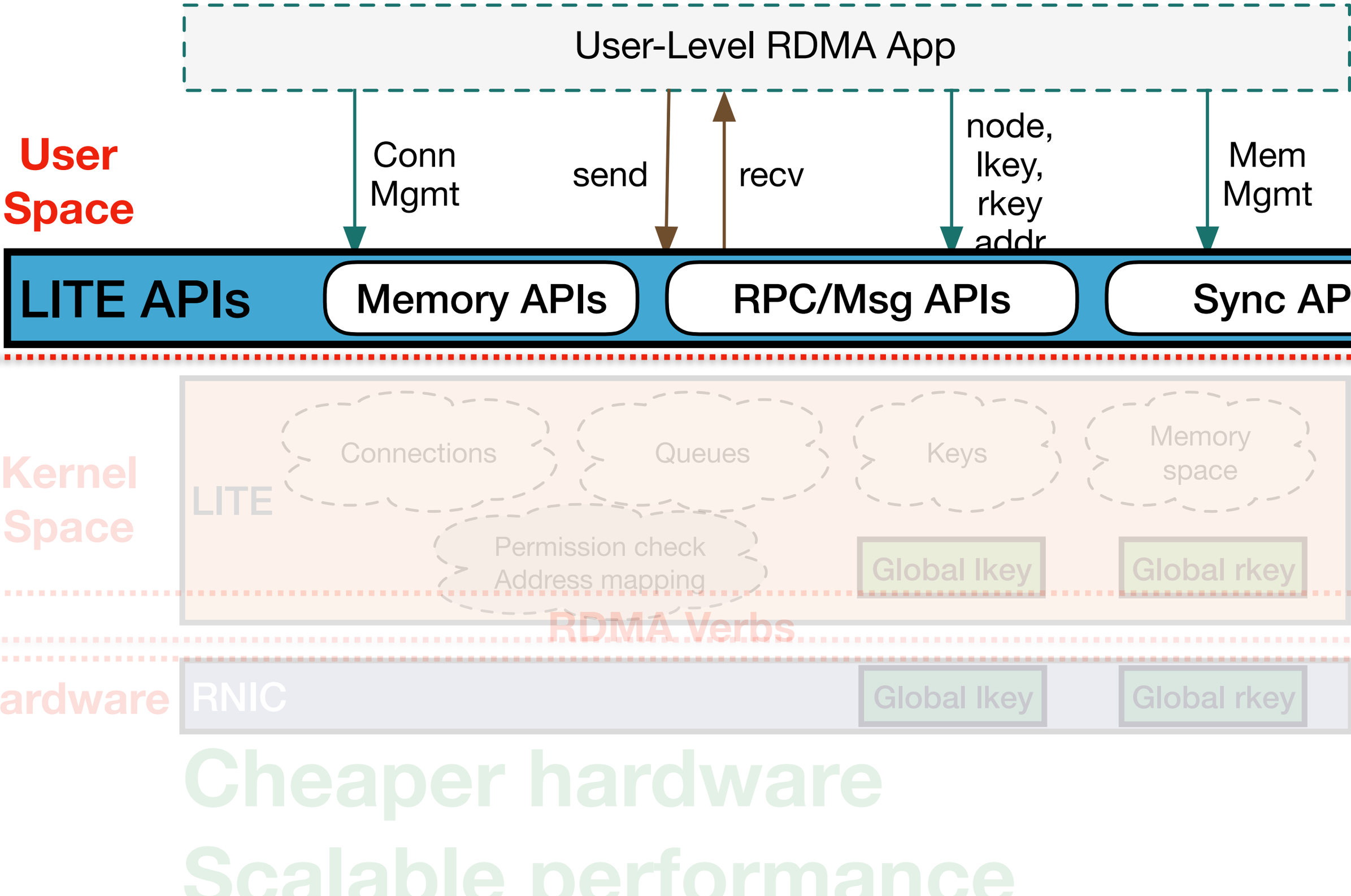
Cheaper hardware
Scalable performance

Simpler applications



Cheaper hardware
Scalable performance

Simpler applications



Cheaper hardware

Scalable performance

Implementing Remote *memset*

Native RDMA

```
struct pingpong_context *ctx;
ctx = calloc(1, sizeof *ctx);

ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0x7b + is_server, size);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;
ctx->pd = ibv_alloc_pd(ctx->context);
ctx->mr = ibv_reg_mr(ctx->pd, ctx->buf, size,IBV_ACCESS_LOCAL_WRITE|IBV_ACCESS_REMOTE_WRITE);
ctx->cq = ibv_create_cq(ctx->context, rx_depth + 1, NULL, ctx->channel, 0);

/* bunch of QP setup .... 50 LOCs */
ctx->qp = ibv_create_qp(ctx->pd, &attr);
ibv_modify_qp(ctx->qp, &attr,IBV_QP_STATE|IBV_QP_PKEY_INDEX|IBV_QP_PORT|IBV_QP_ACCESS_FLAGS);
/* build connections .... 100 LOCs */
/* exchange all required information, qpns, psns, and keys */

/* start doing write request */
struct ibv_sge sg;
struct ibv_send_wr wr;
struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request*/
sg.addr = (uintptr_t)buf_addr;
sg.length = buf_size;
sg.lkey = ctx->mr->lkey;

wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_sge = 1;
wr.opcode  = IBV_WR_RDMA_READ;
wr.send_flags = IBV_SEND_SIGNALED;
wr.wr.rdma.remote_addr = remote_address
wr.wr.rdma.rkey = remote_key;

/* send out the request */
ibv_post_send(qp, &wr, &bad_wr);
struct ibv_wc wc[2];
ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */

return ctx;
```

Implementing Remote *memset*

Native RDMA

```
1   struct pingpong_context *ctx;
2   ctx = calloc(1, sizeof *ctx);
3
4   ctx->size = size;
5   ctx->rx_depth = rx_depth;
6   ctx->buf = malloc(roundup(size, page_size));
7   memset(ctx->buf, 0x7b + is_server, size);
8   ctx->context = ibv_open_device(ib_dev);
9   ctx->channel = NULL;
10  ctx->pd = ibv_alloc_pd(ctx->context);
11  ctx->mr = ibv_reg_mr(ctx->pd, ctx->buf, size,IBV_ACCESS_LOCAL_WRITE|IBV_ACCESS_REMOTE_WRITE);
12  ctx->cq = ibv_create_cq(ctx->context, rx_depth + 1, NULL, ctx->channel, 0);
13
14  /* bunch of QP setup .... 50 LOCs */
15  ctx->qp = ibv_create_qp(ctx->pd, &attr);
16  ibv_modify_qp(ctx->qp, &attr,IBV_QP_STATE|IBV_QP_PKEY_INDEX|IBV_QP_PORT|IBV_QP_ACCESS_FLAGS);
17  /* build connections .... 100 LOCs */
18  /* exchange all required information, qpns, psns, and keys */
19
20  /* start doing write request */
21  struct ibv_sge sg;
22  struct ibv_send_wr wr;
23  struct ibv_send_wr *bad_wr;
24  memset(&wr, 0, sizeof(wr));
25  memset(&sg, 0, sizeof(sg));
26
27  /* setup all required metadata for a write request*/
28  sg.addr = (uintptr_t)buf_addr;
29  sg.length = buf_size;
30  sg.lkey = ctx->mr->lkey;
31
32  wr.wr_id = 0;
33  wr.sg_list = &sg;
34  wr.num_sge = 1;
35  wr.opcode  = IBV_WR_RDMA_READ;
36  wr.send_flags = IBV_SEND_SIGNALED;
37  wr.wr.rdma.remote_addr = remote_address
38  wr.wr.rdma.rkey = remote_key;
39
40  /* send out the request */
41  ibv_post_send(qp, &wr, &bad_wr);
42  struct ibv_wc wc[2];
43  ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */
44
45  return ctx;
```

LITE

```
1  LITE_join(IP);
2  uin64_t lh = LITE_malloc(node, size);
3  LITE_memset(lh, 0, offset, size);
```

Implementing Remote *memset*

Native RDMA

```
1   struct pingpong_context *ctx;
2   ctx = calloc(1, sizeof *ctx);
3
4   ctx->size = size;
5   ctx->rx_depth = rx_depth;
6   ctx->buf = malloc(roundup(size, page_size));
7   memset(ctx->buf, 0x7b + is_server, size);
8   ctx->context = ibv_open_device(ib_dev);
9   ctx->channel = NULL;
```

```
23  struct ibv_send_wr *bad_wr;
24  memset(&wr, 0, sizeof(wr));
25  memset(&sg, 0, sizeof(sg));
26
27  /* setup all required metadata for a write request*/
28  sg.addr = (uintptr_t)buf_addr;
29  sg.length = buf_size;
30  sg.lkey = ctx->mr->lkey;
31
32  wr.wr_id = 0;
33  wr.sg_list = &sg;
34  wr.num_sge = 1;
35  wr.opcode  = IBV_WR_RDMA_READ;
36  wr.send_flags = IBV_SEND_SIGNALED;
37  wr.wr.rdma.remote_addr = remote_address
38  wr.wr.rdma.rkey = remote_key;
39
40  /* send out the request */
41  ibv_post_send(qp, &wr, &bad_wr);
42  struct ibv_wc wc[2];
43  ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */
44
45  return ctx;
```

LITE

```
1  LITE_join(IP);
2  uin64_t lh = LITE_malloc(node, size);
3  LITE_memset(lh, 0, offset, size);
```

Implementing Remote *memset*

Native RDMA

```c
struct pingpong_context *ctx;
ctx = calloc(1, sizeof *ctx);

ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0x7b + is_server, size);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;
```

```c
struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request*/
sg.addr = (uintptr_t)buf_addr;
sg.length = buf_size;
sg.lkey = ctx->mr->lkey;

wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_sge = 1;
wr.opcode  = IBV_WR_RDMA_READ;
wr.send_flags = IBV_SEND_SIGNALED;
wr.wr.rdma.remote_addr = remote_address
wr.wr.rdma.rkey = remote_key;

/* send out the request */
ibv_post_send(qp, &wr, &bad_wr);
struct ibv_wc wc[2];
ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */

return ctx;
```

LITE

```c
LITE_join(IP);
uin64_t lh = LITE_malloc(node, size);
LITE_memset(lh, 0, offset, size);
```

All problems in computer science can be solved by another level of indirection

Butler Lampson

All problems in computer science can be solved by another level of indirection

Butler Lampson

David Wheeler

All problems in computer science can be solved by another level of indirection

Butler Lampson

David Wheeler

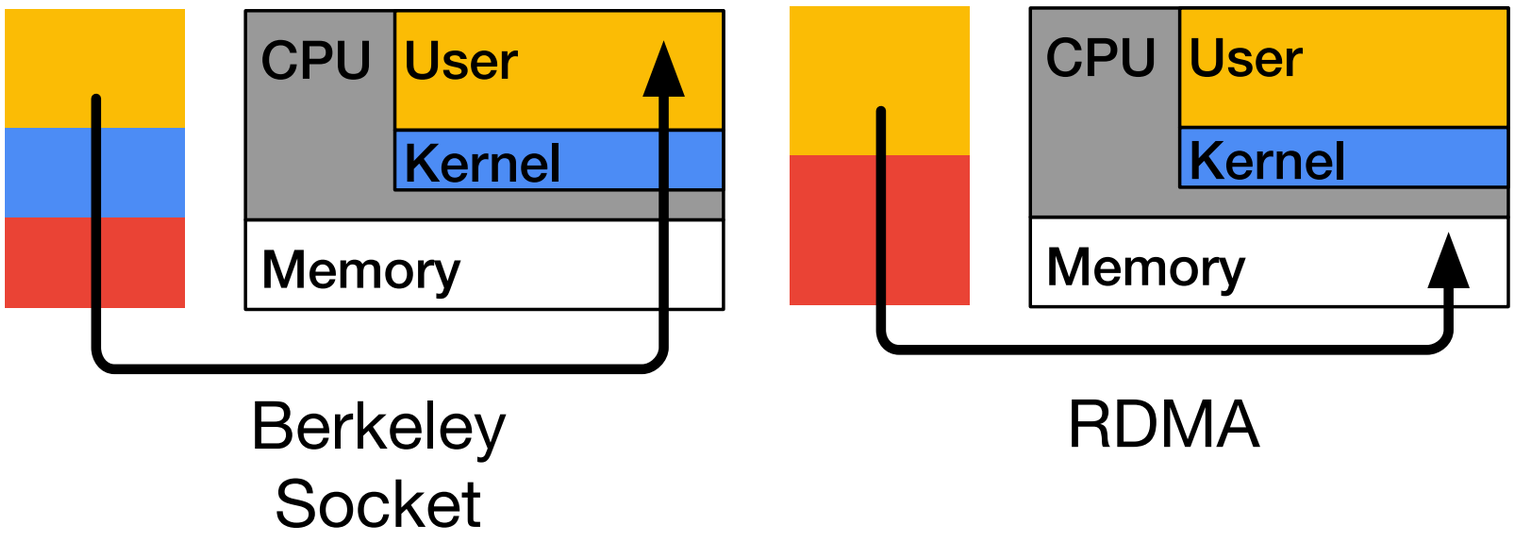
except for the problem of too many layers of indirection

– David Wheeler

Main Challenge: How to preserve the performance benefit of RDMA?

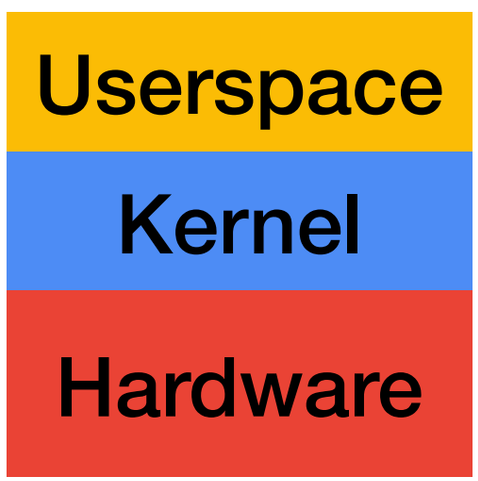
Design Principles

1.Indirection only at local for one-sided RDMA



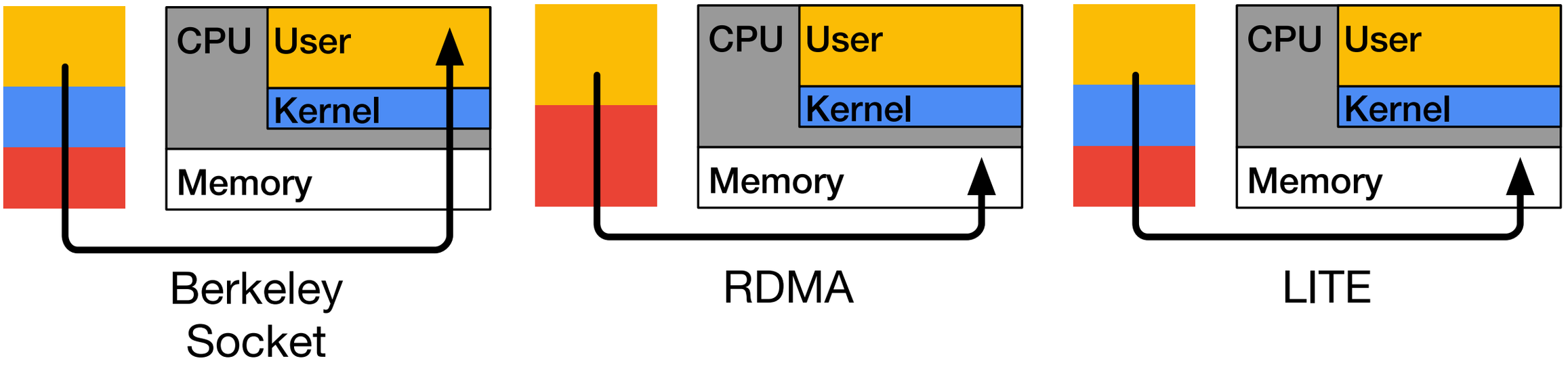
Berkeley Socket

RDMA



Design Principles

1.Indirection only at local for one-sided RDMA



Berkeley Socket

RDMA

LITE

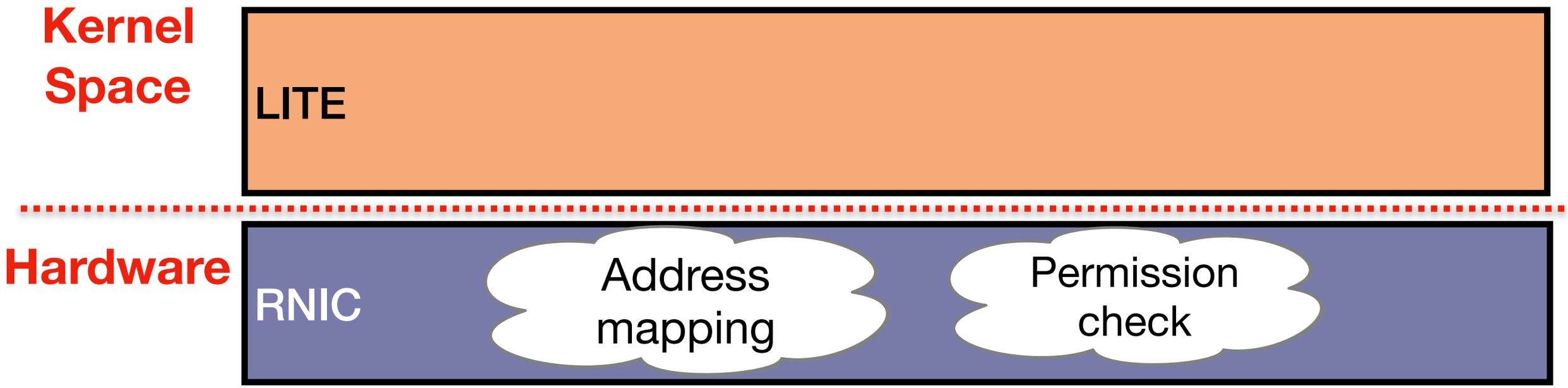
Userspace

Kernel

Hardware

Design Principles

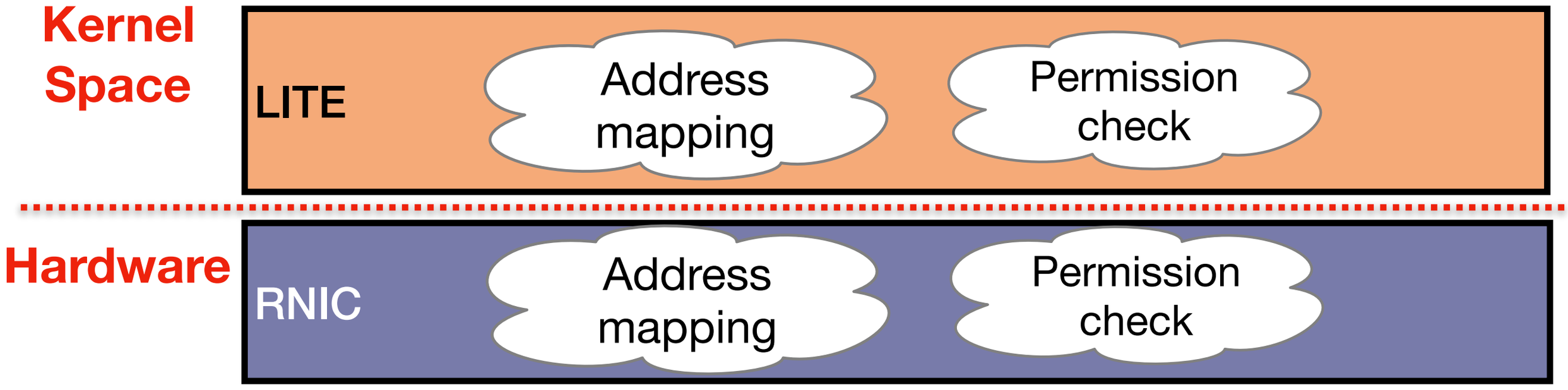
1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection



Design Principles

1. Indirection only at local for one-sided RDMA

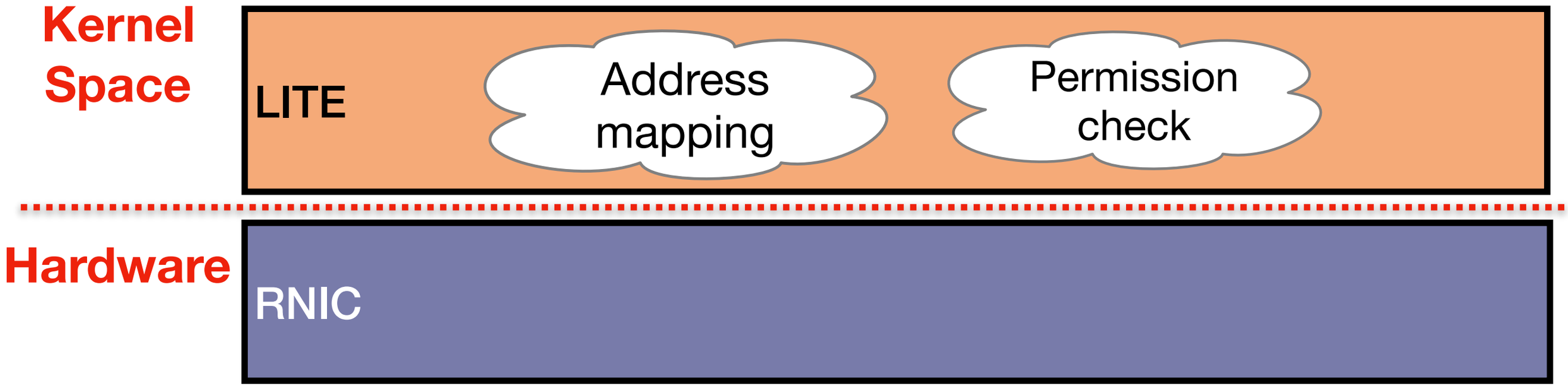
2. Avoid hardware indirection



Design Principles

1. Indirection only at local for one-sided RDMA

2. Avoid hardware indirection



No redundant indirection
Scalable performance

Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost

Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost

~~except for the problem of too many layers of indirection – David Wheeler~~

Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost

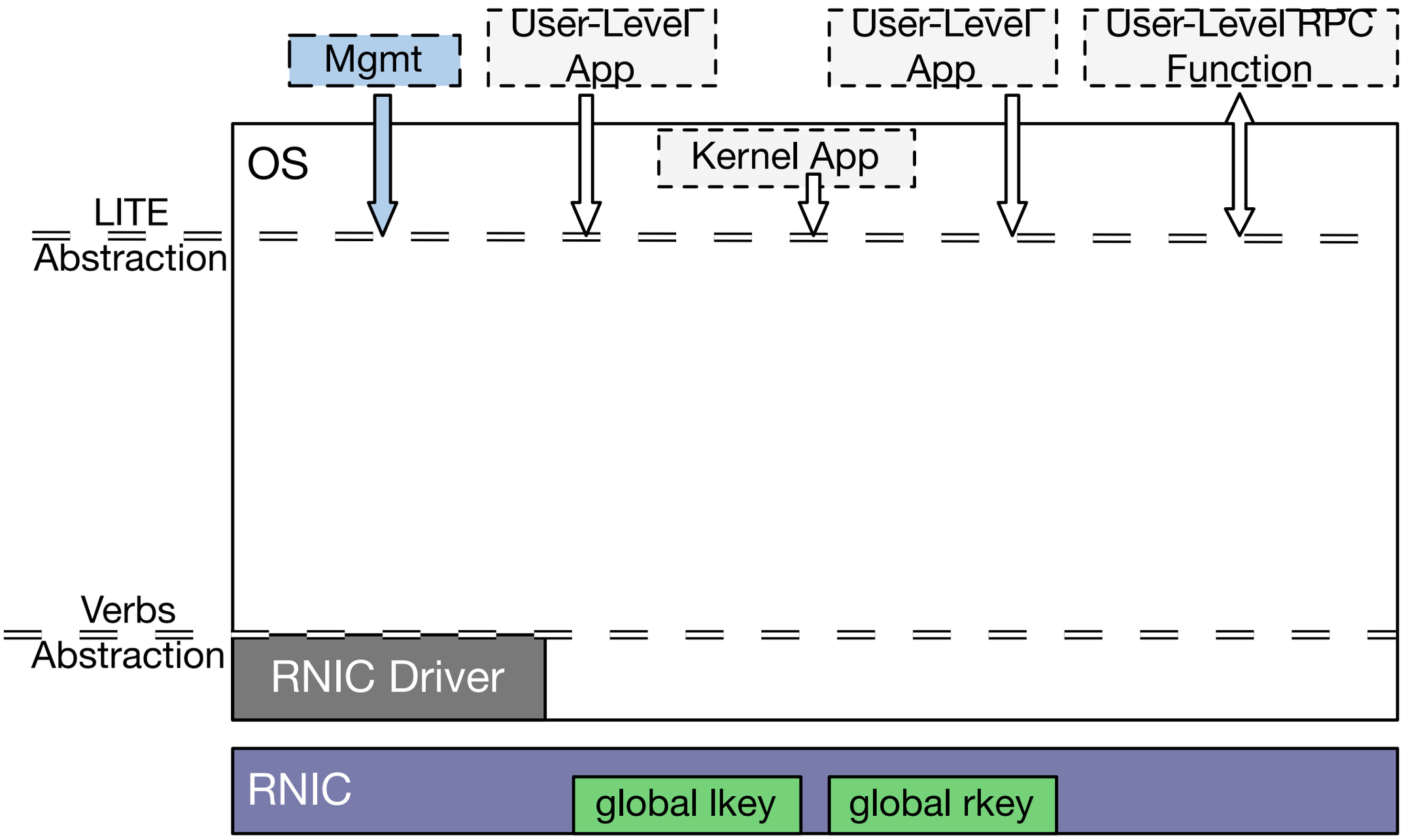
~~except for the problem of too many layers of indirection – David Wheeler~~

Great Performance and Scalability

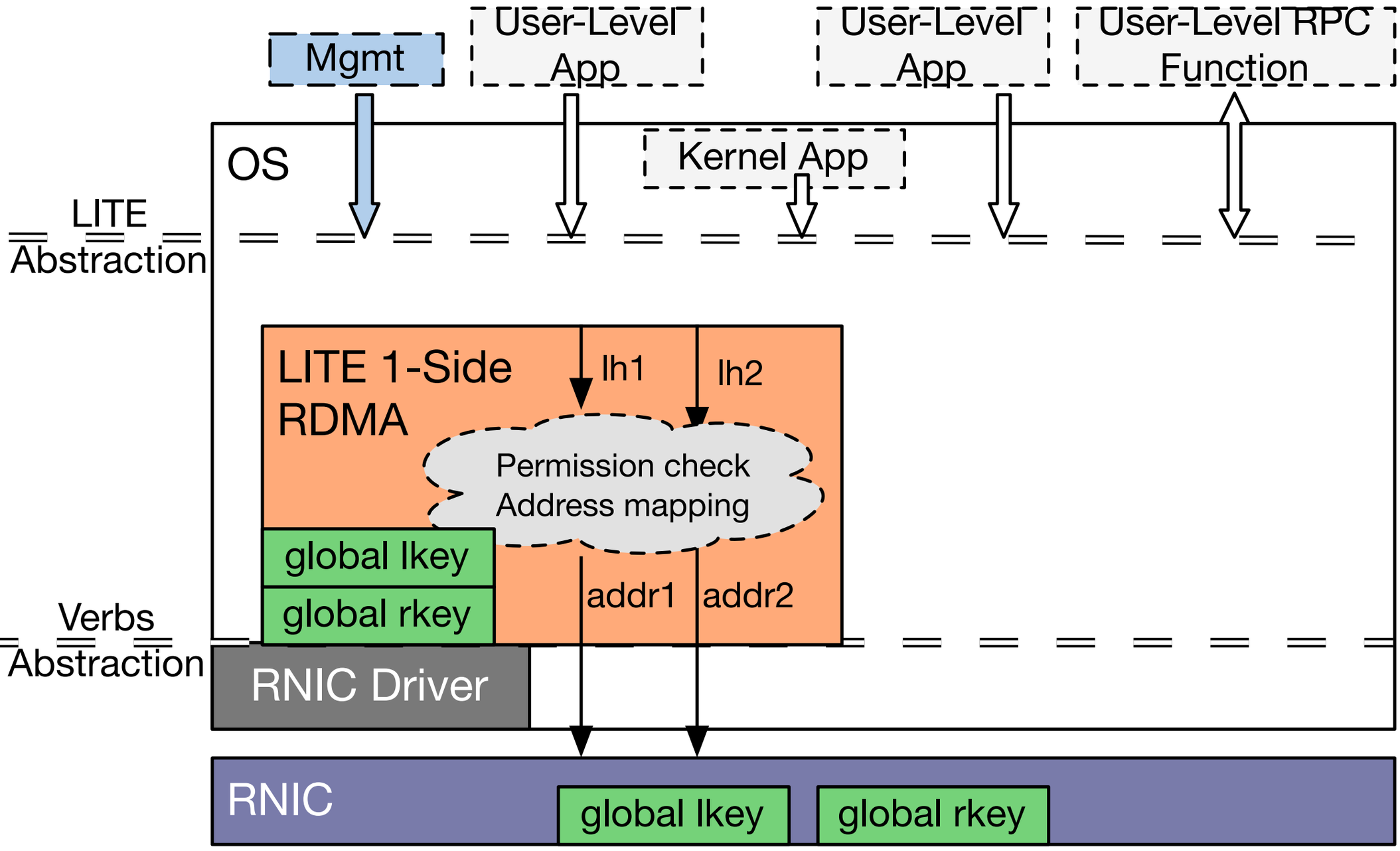
Outline

- Introduction and motivation
- Overall design and abstraction
- LITE internals
- LITE applications
- Conclusion

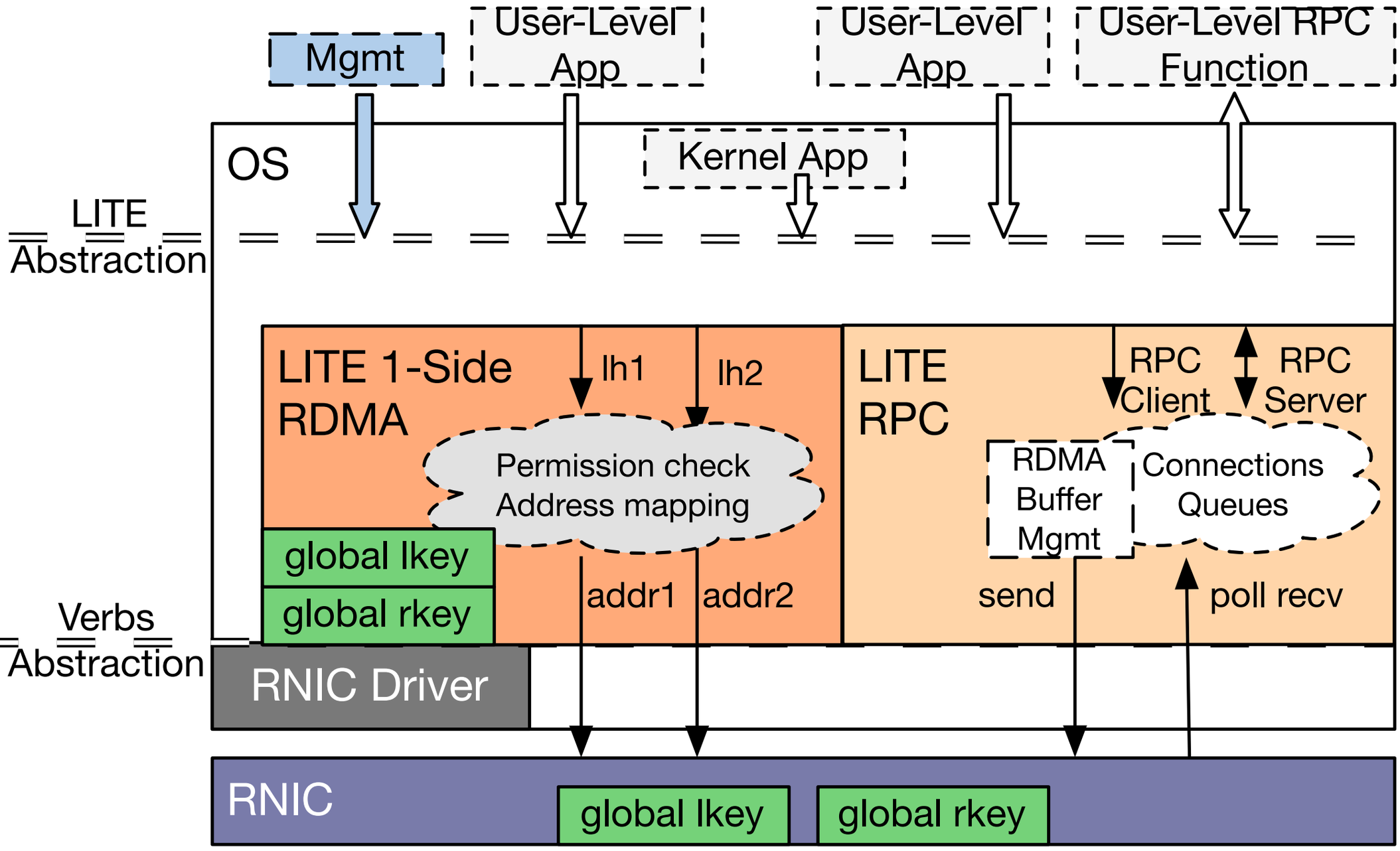
LITE - Architecture



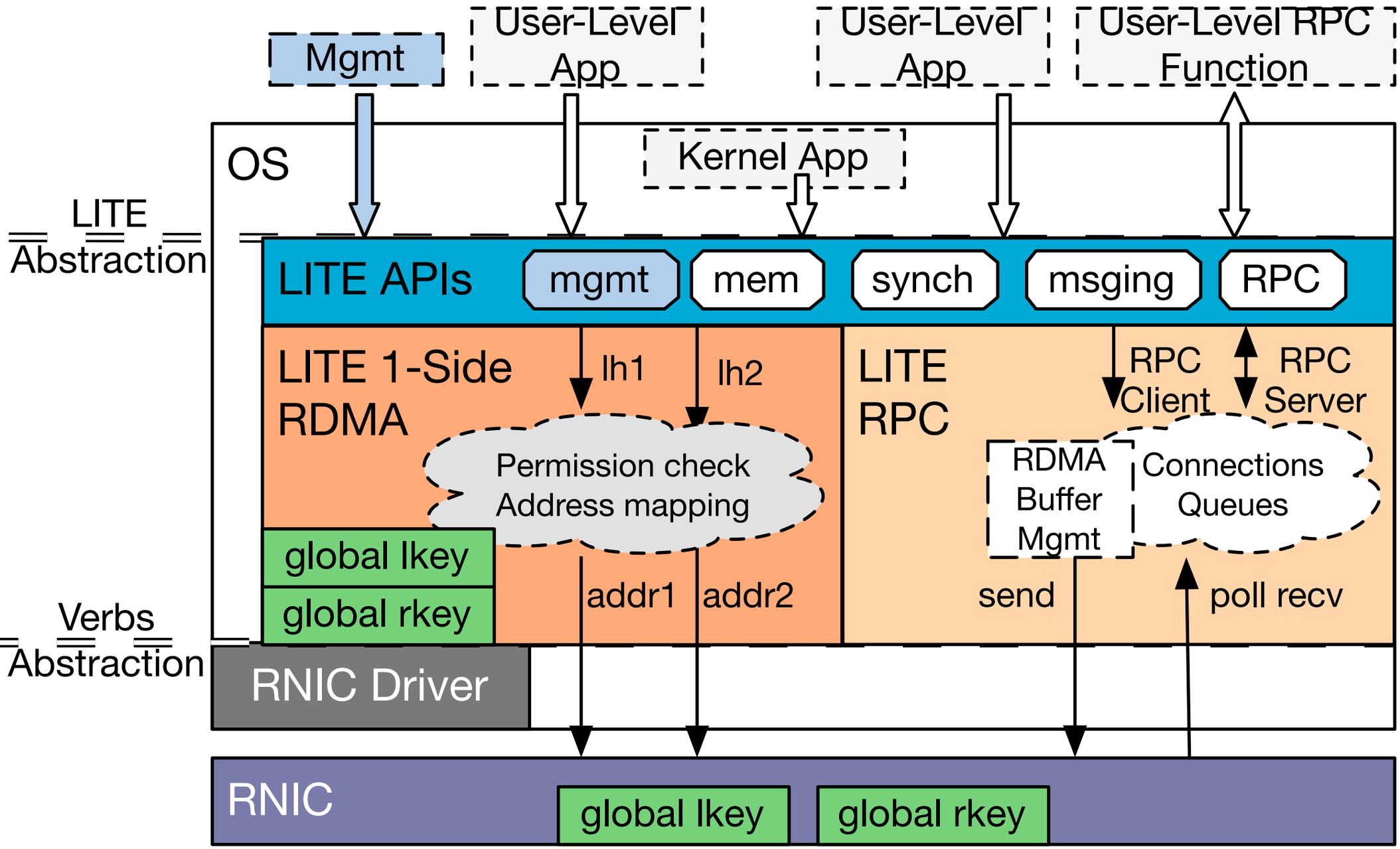
LITE - Architecture



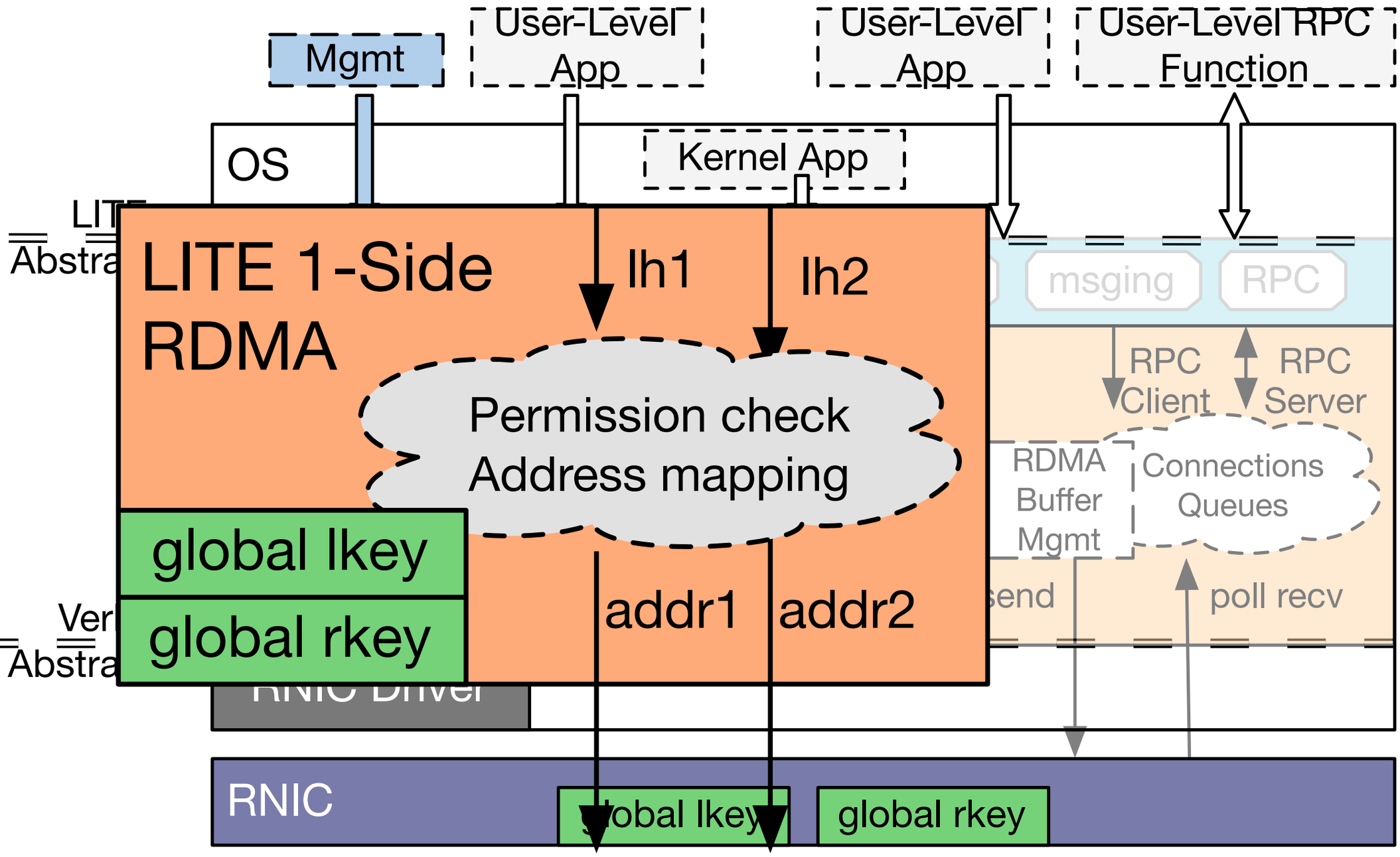
LITE - Architecture



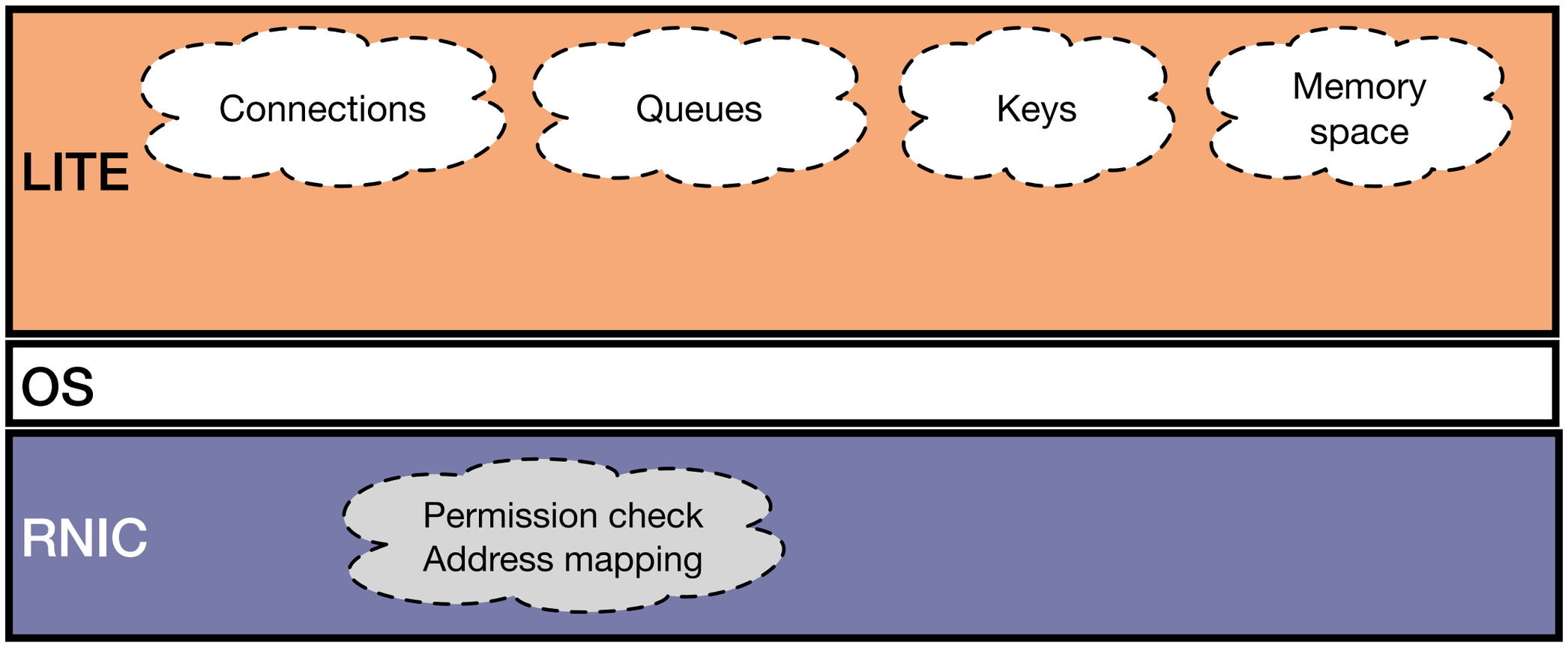
LITE - Architecture



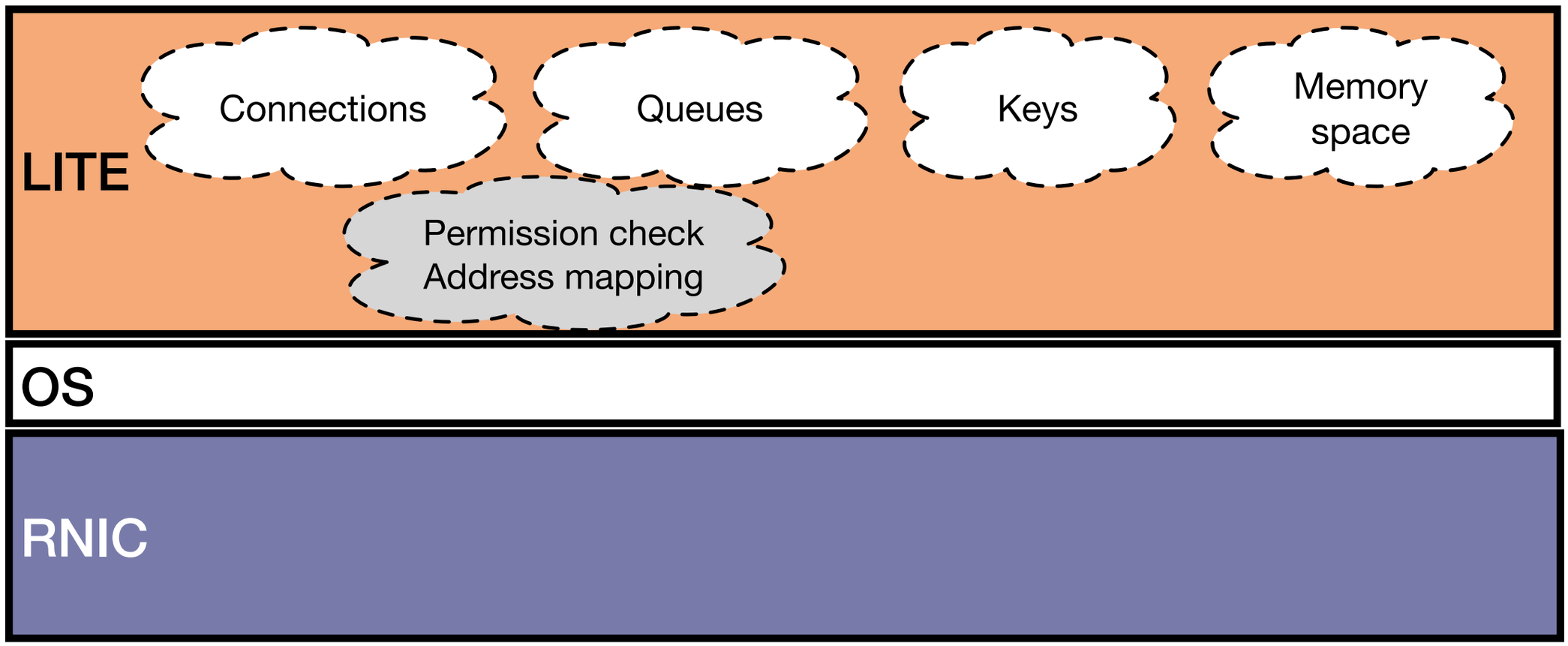
LITE - Architecture



Onload Costly Operations

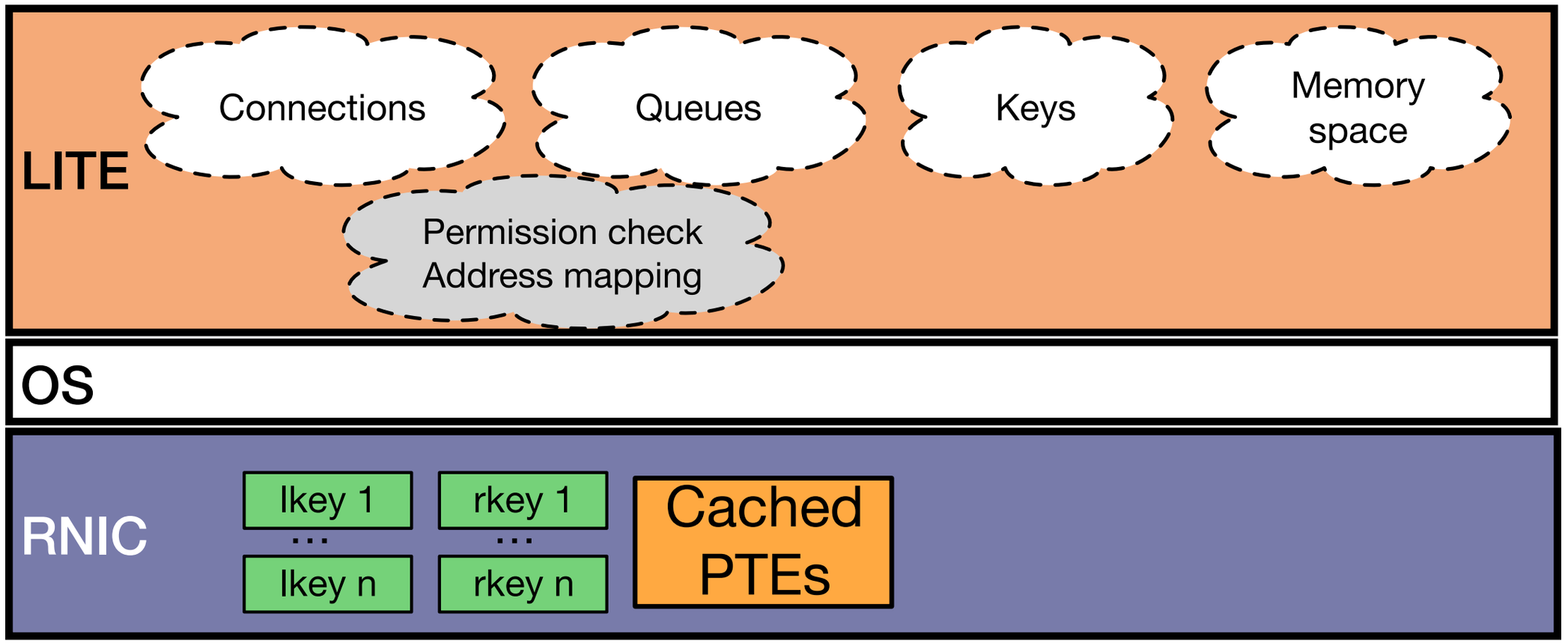


Onload Costly Operations



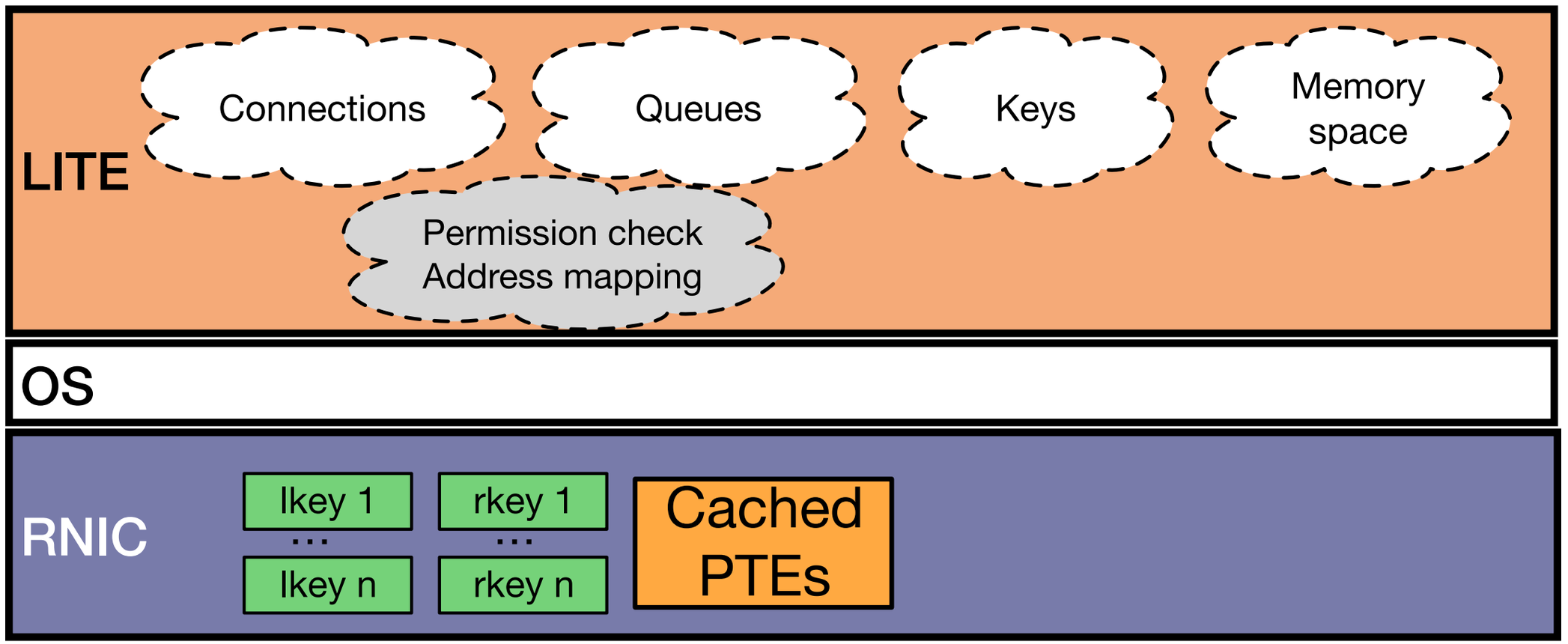
Perform **address mapping** and **protection** in kernel

Avoid Hardware Indirection



Challenge: *How to eliminate hardware indirection* ***without changing hardware****?*

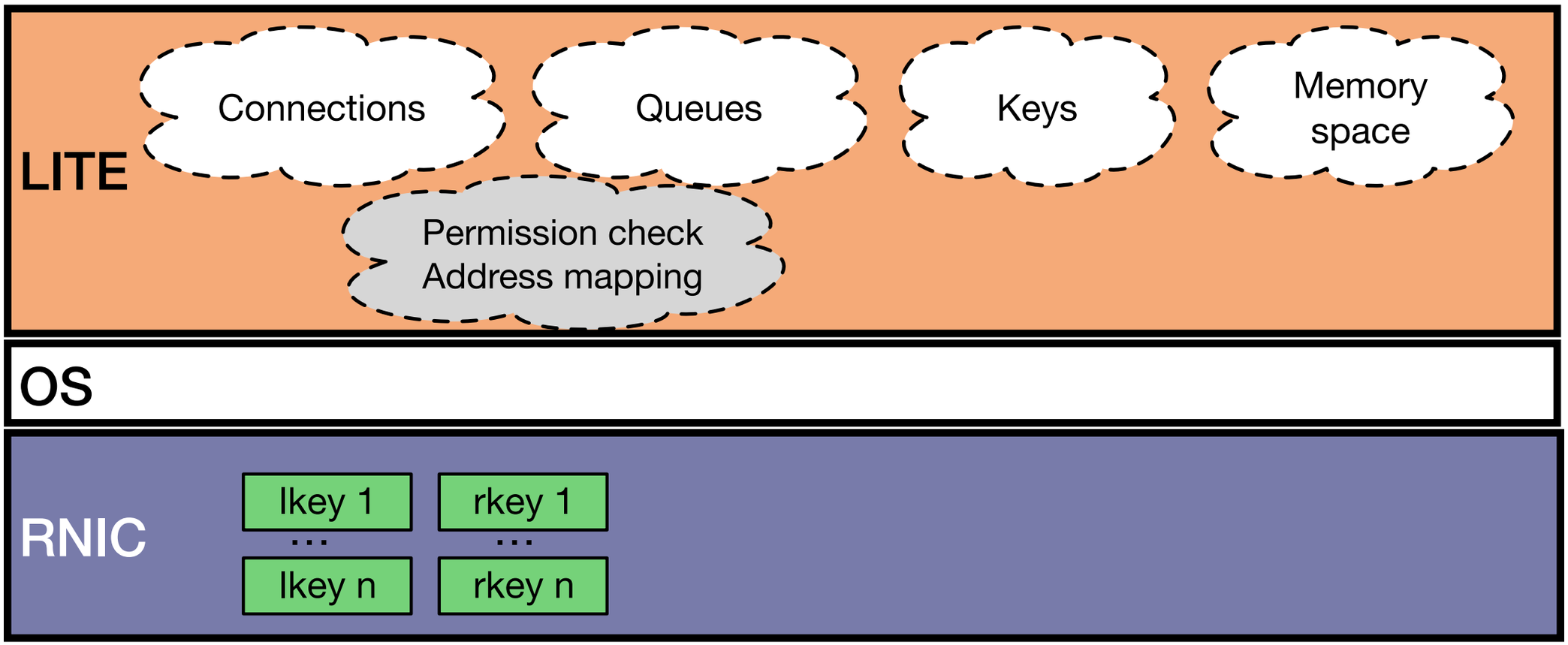
Avoid Hardware Indirection



Challenge: *How to eliminate hardware indirection* ***without changing hardware****?*

- Register with **physical address** → no need for any PTEs

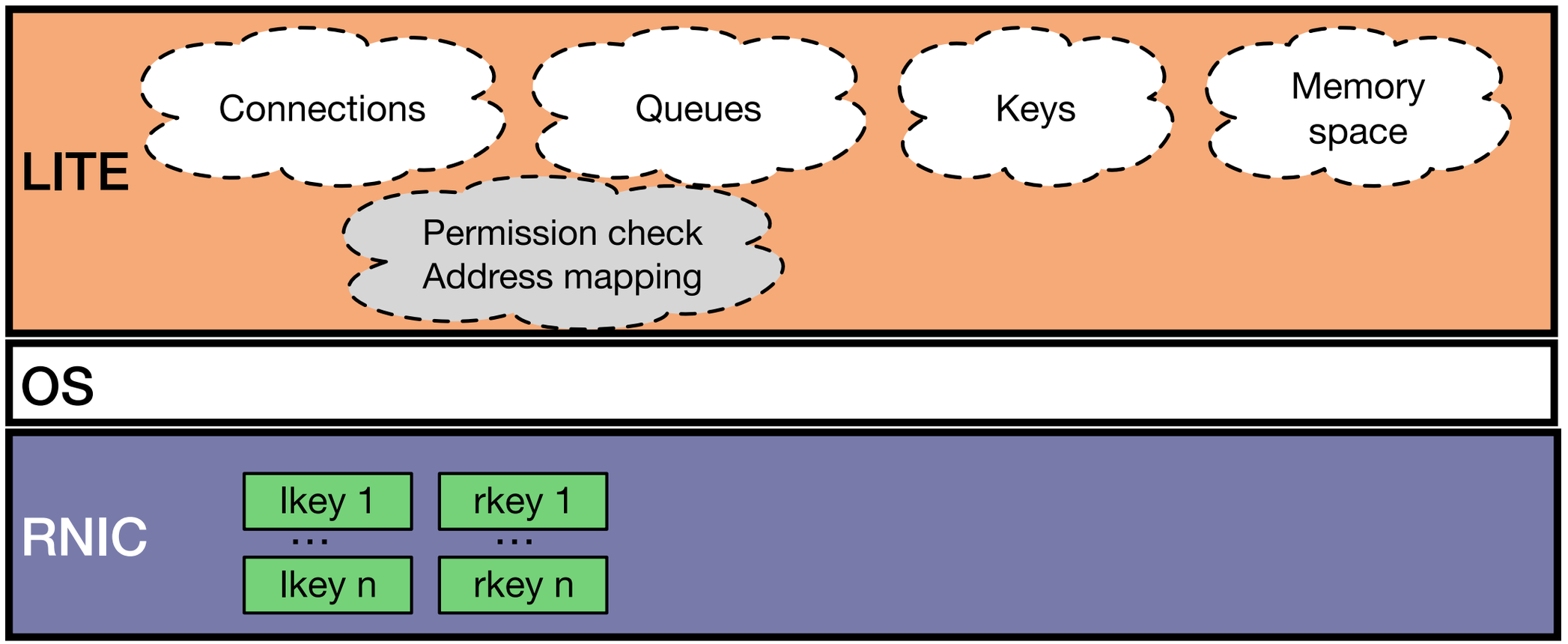
Avoid Hardware Indirection



Challenge: *How to eliminate hardware indirection* ***without changing hardware****?*

- Register with **physical address** → no need for any PTEs

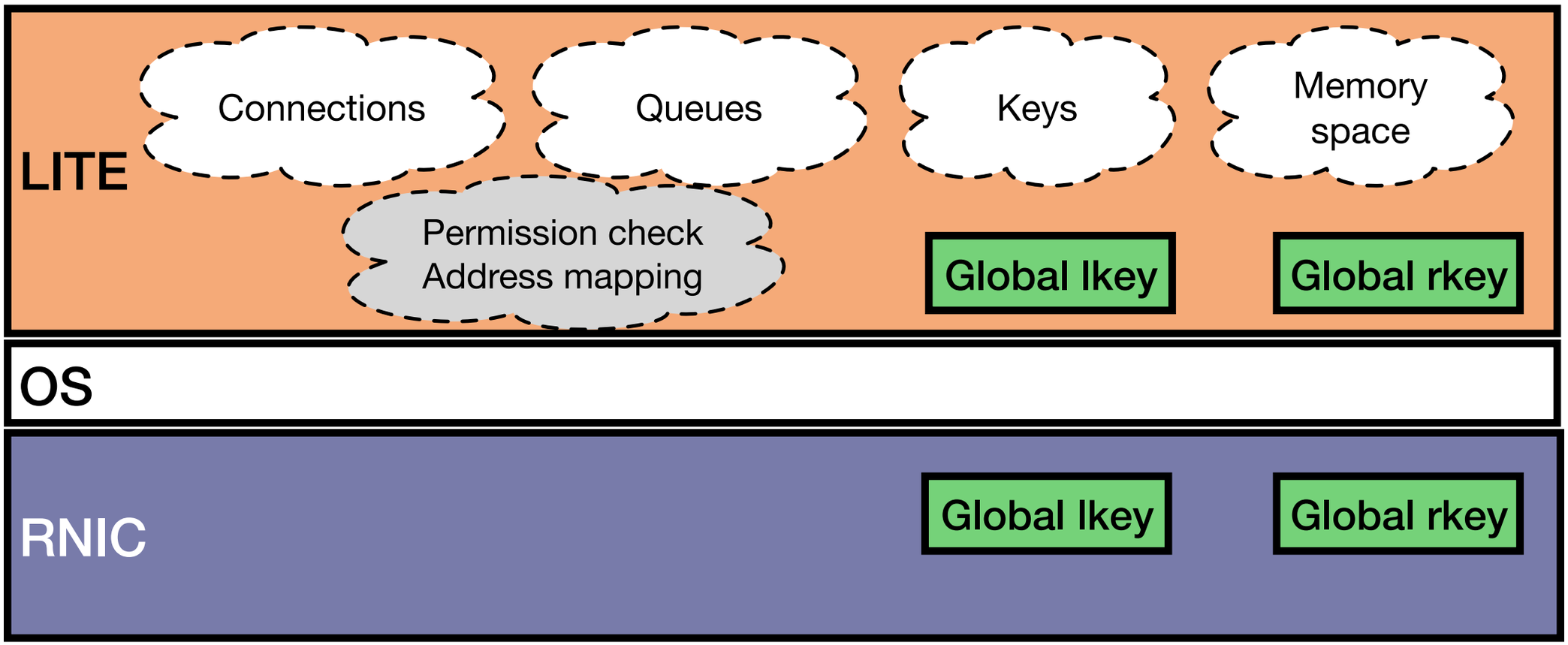
Avoid Hardware Indirection



Challenge: *How to eliminate hardware indirection* ***without changing hardware****?*

- Register with **physical address** → no need for any PTEs
- Register **whole memory** at once → one global key

Avoid Hardware Indirection

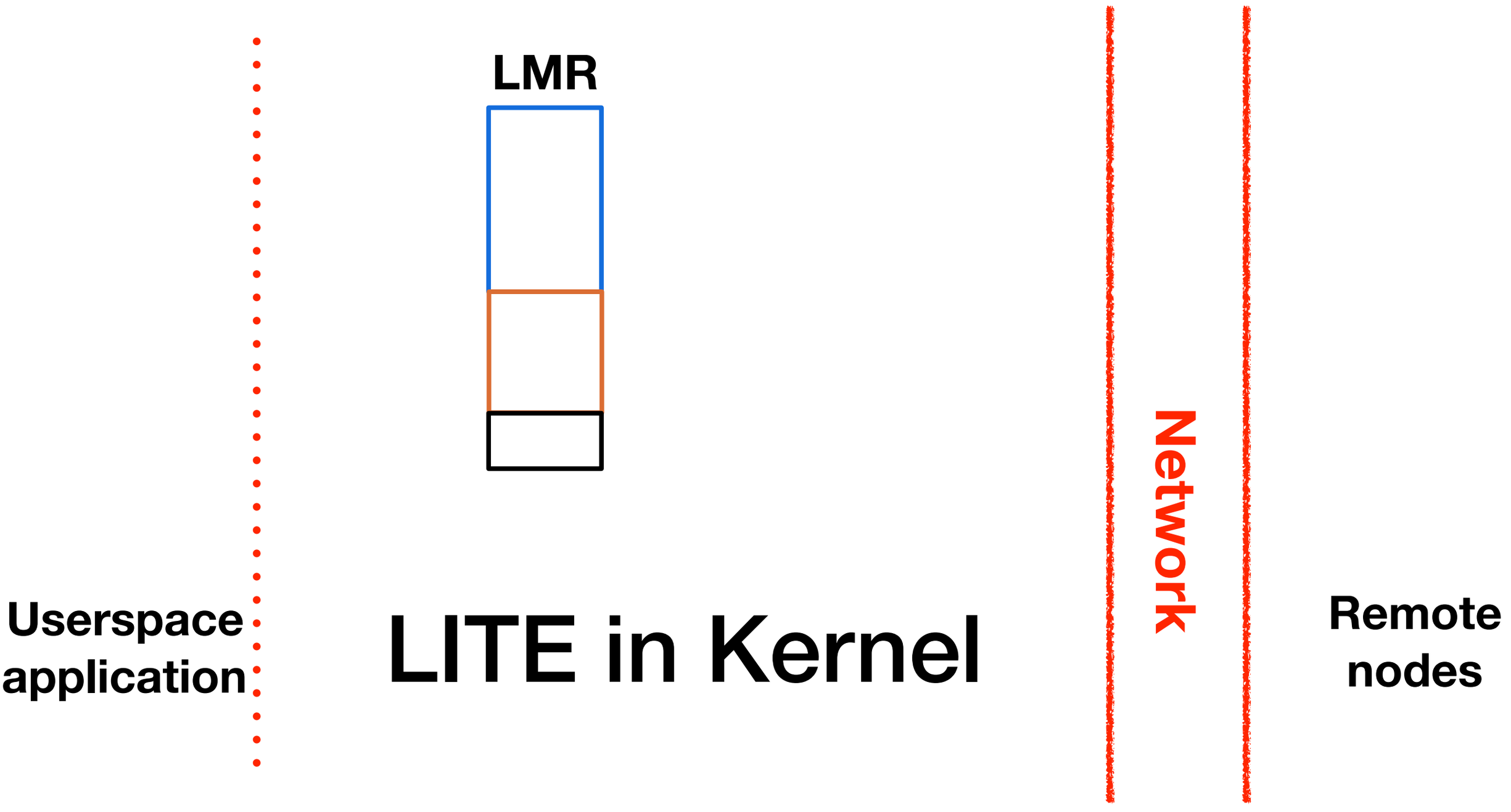


Challenge: *How to eliminate hardware indirection* ***without changing hardware****?*

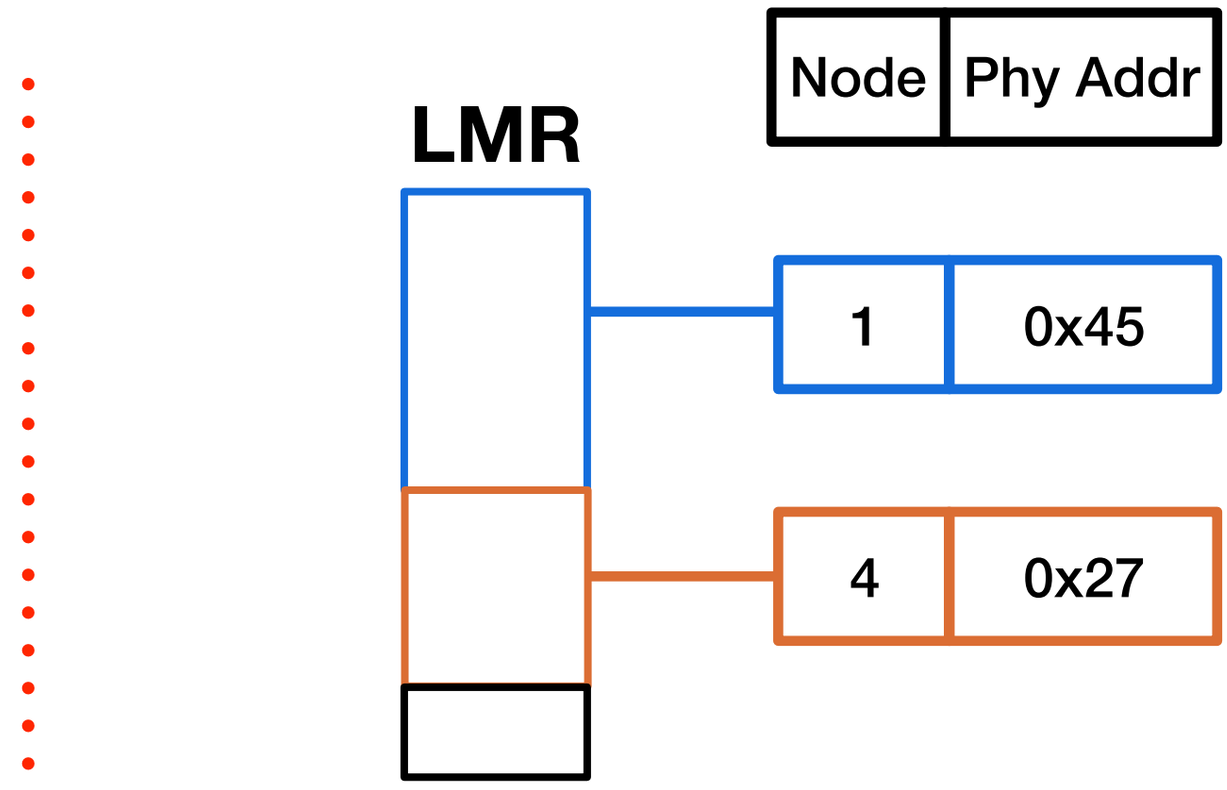
- Register with **physical address** → no need for any PTEs
- Register **whole memory** at once → one global key

LITE LMR and RDMA

LITE LMR and RDMA



LITE LMR and RDMA



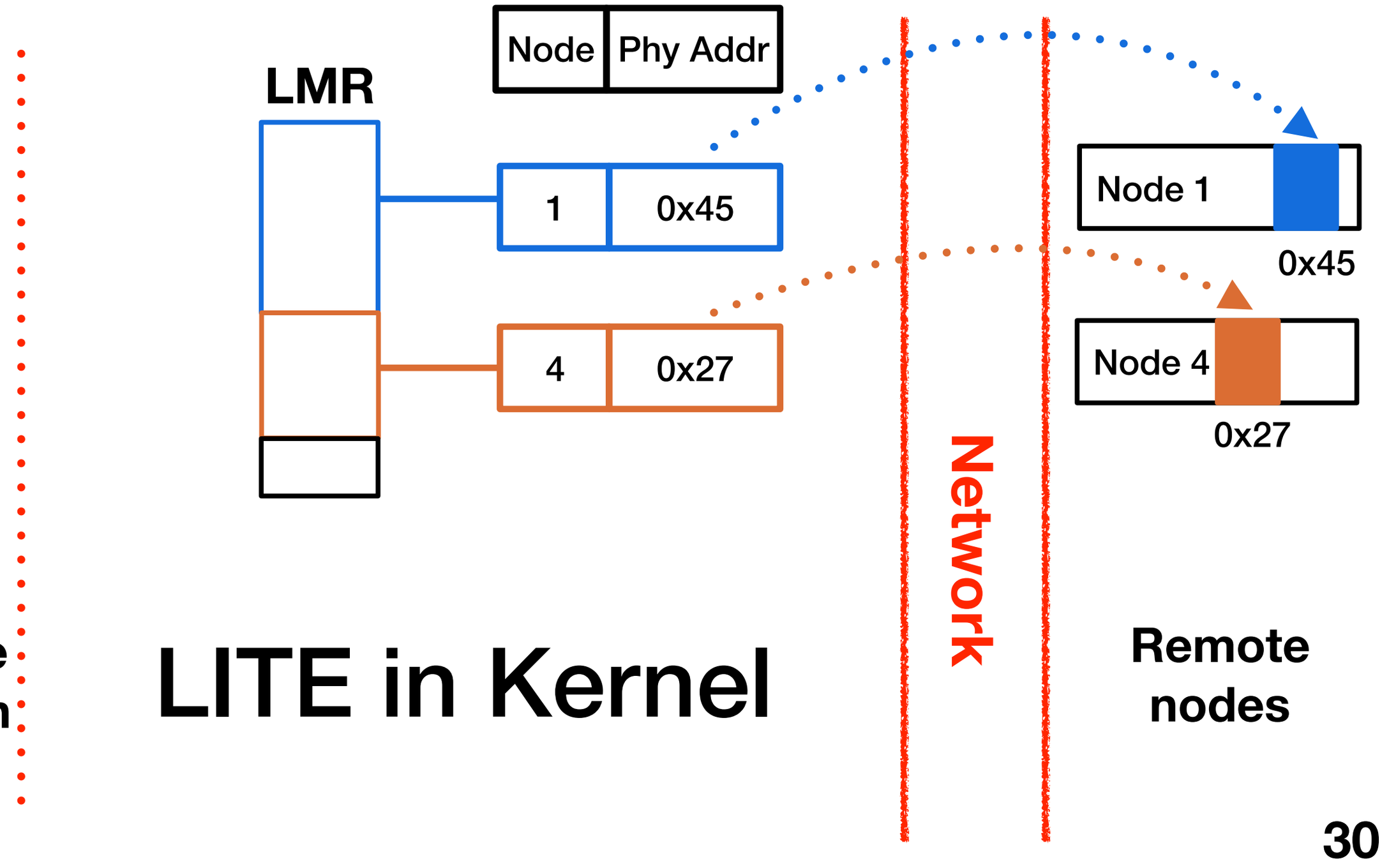
Userspace application

LITE in Kernel

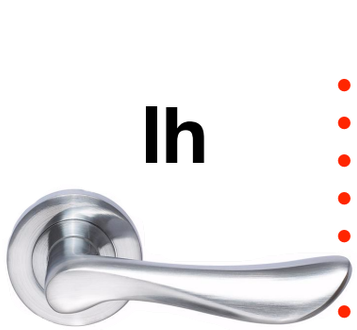
Network

Remote nodes

LITE LMR and RDMA



LITE LMR and RDMA



lh

LMR

Node	Phy Addr
1	0x45
4	0x27

Node 1

0x45

Node 4

0x27

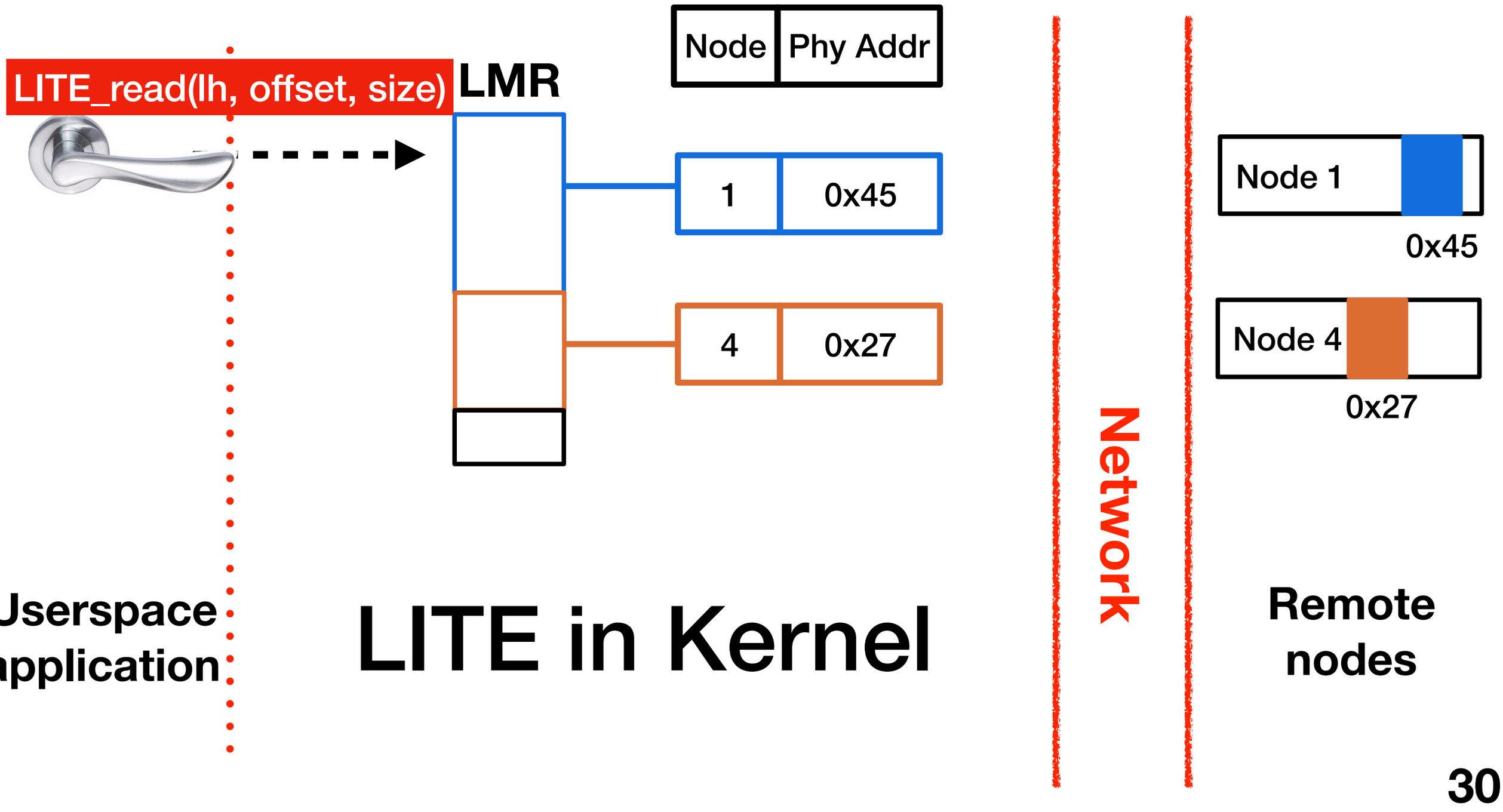
Network

Userspace application

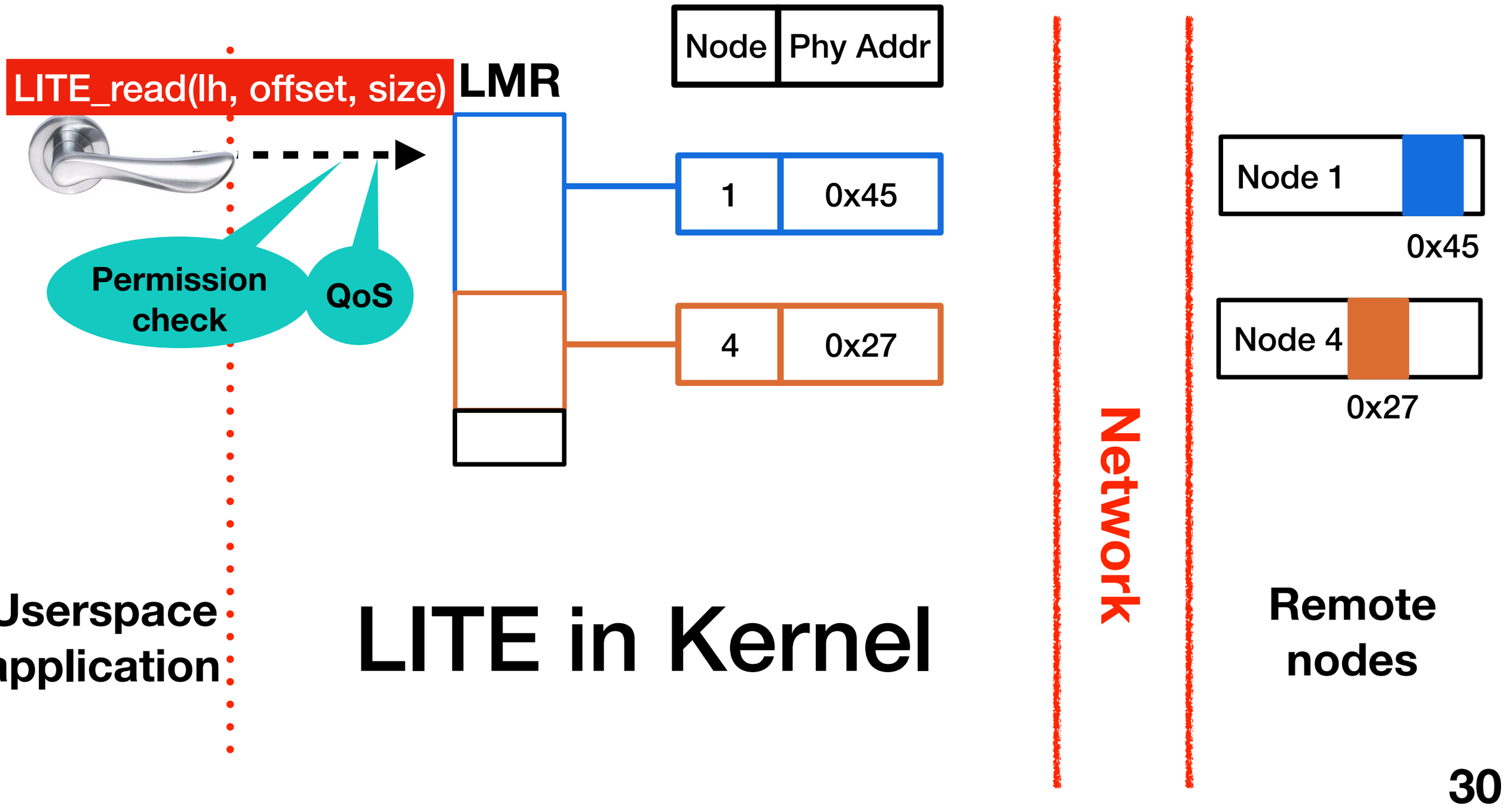
LITE in Kernel

Remote nodes

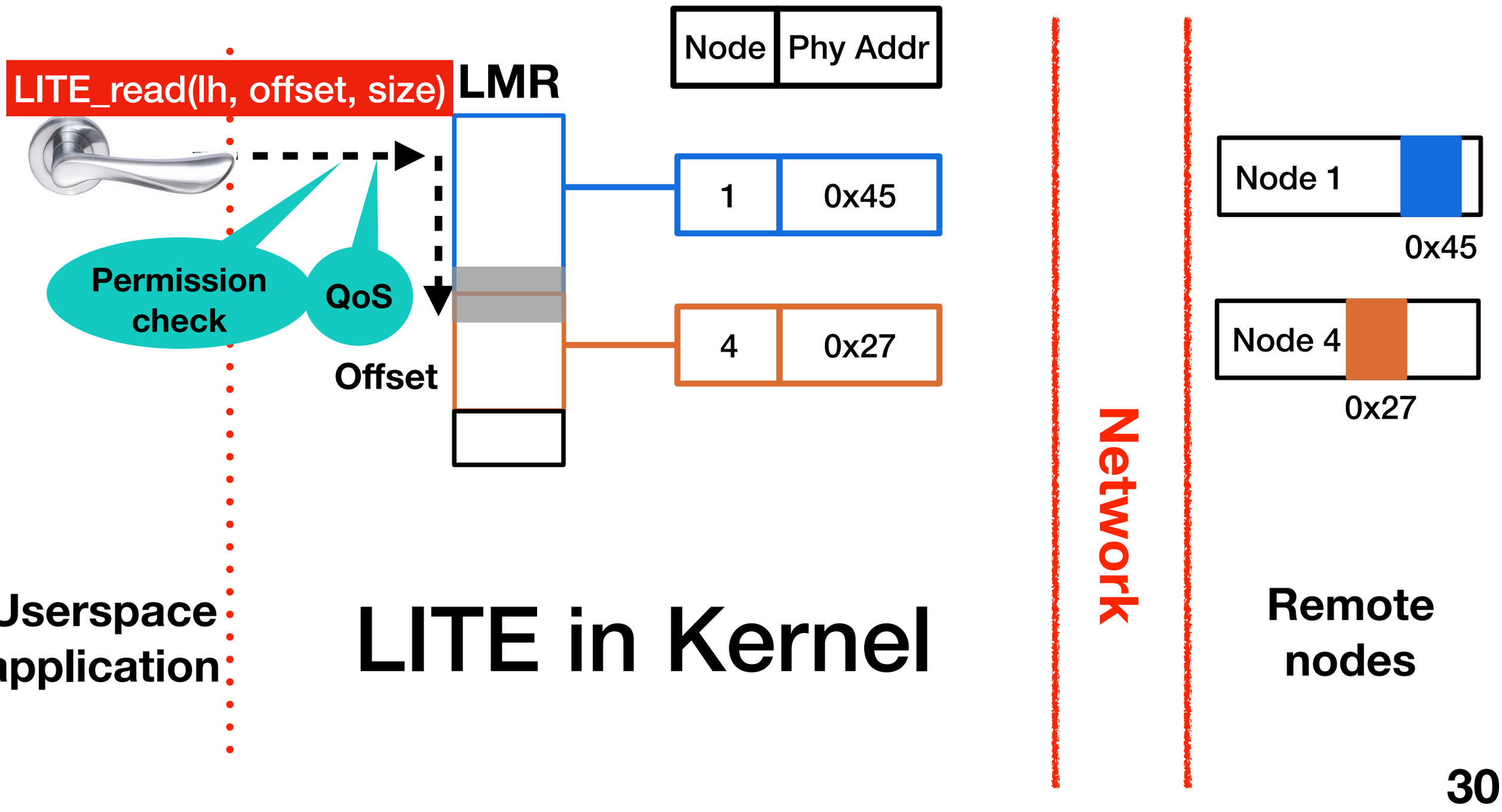
LITE LMR and RDMA



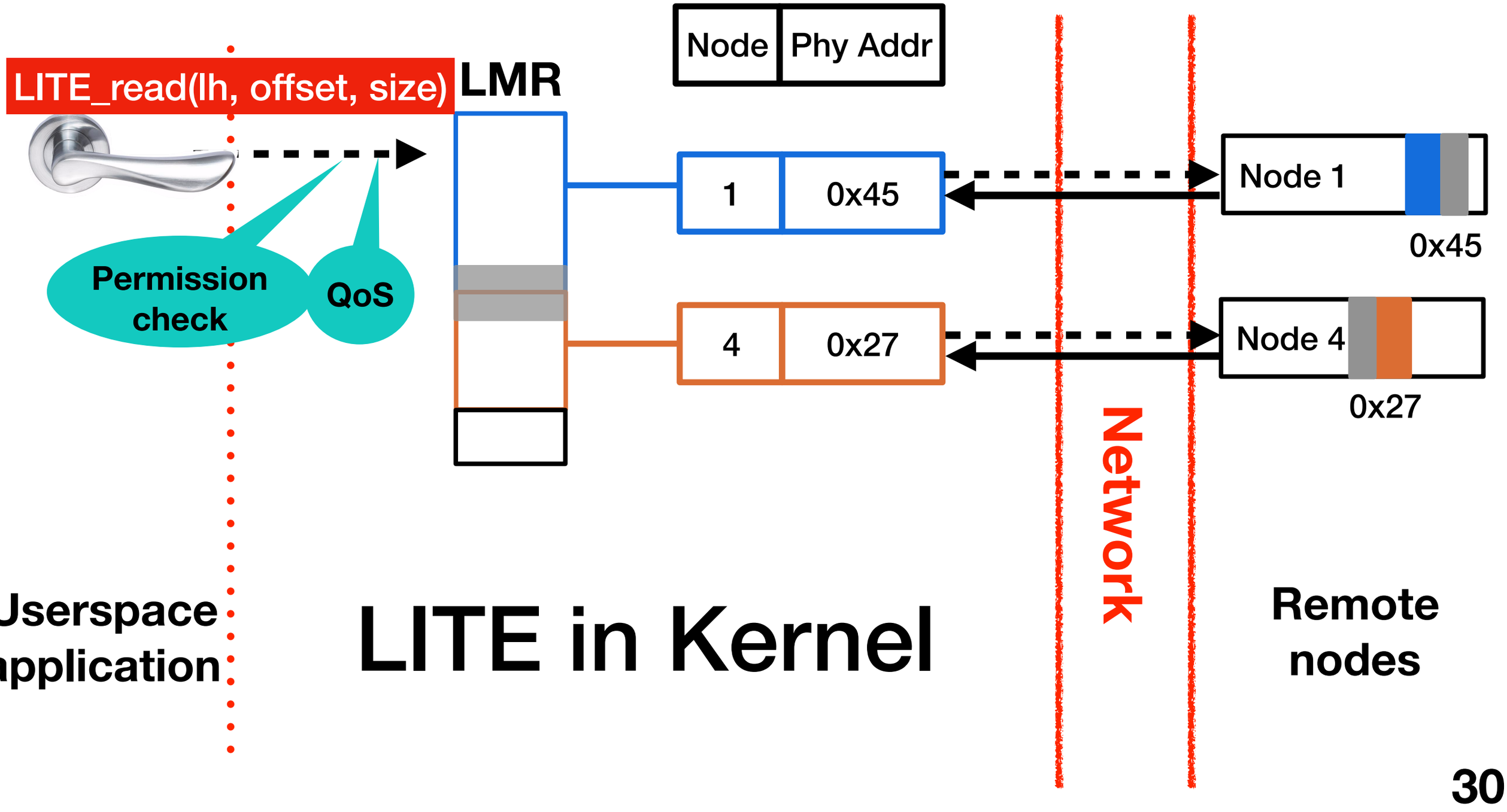
LITE LMR and RDMA



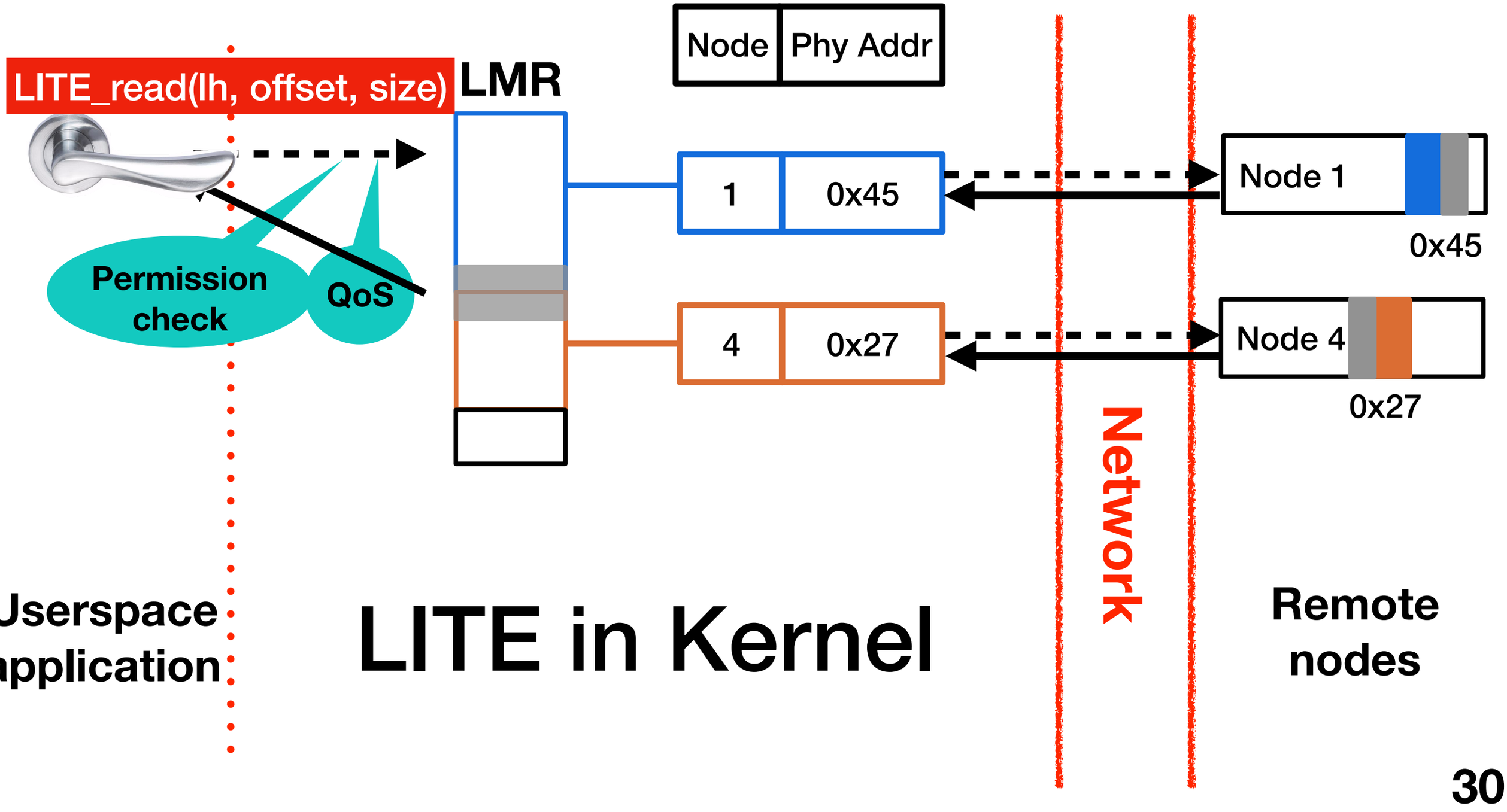
LITE LMR and RDMA



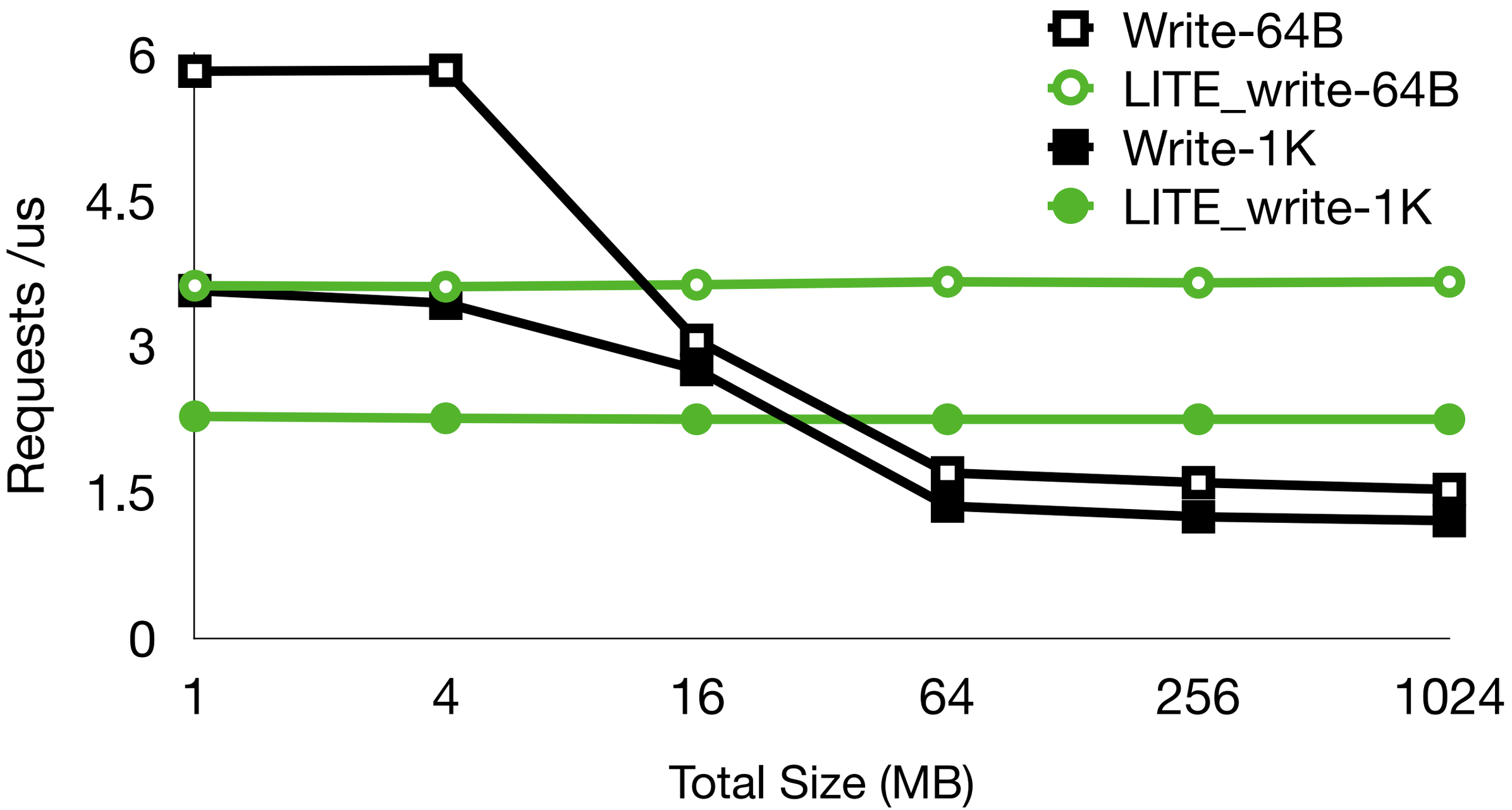
LITE LMR and RDMA



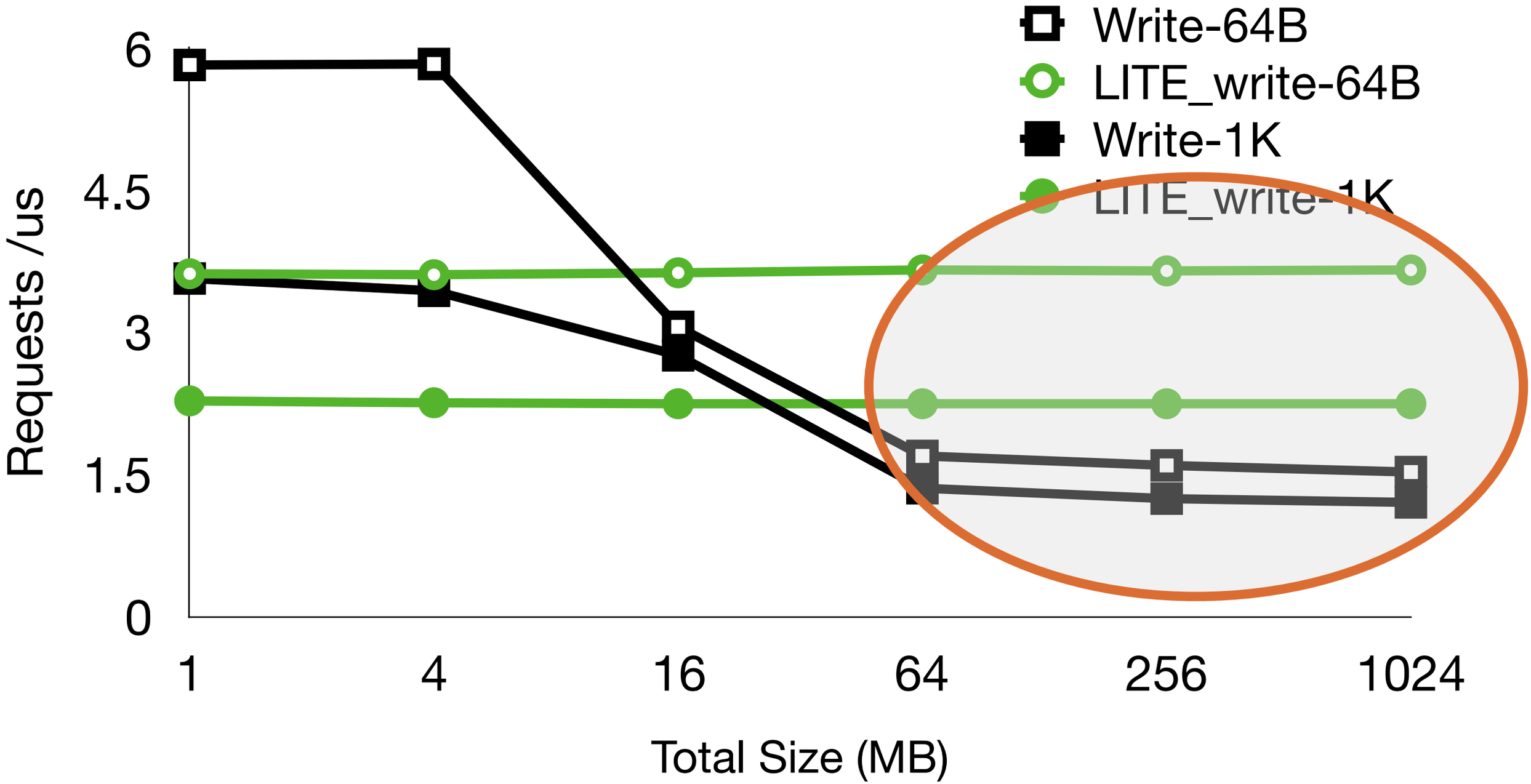
LITE LMR and RDMA



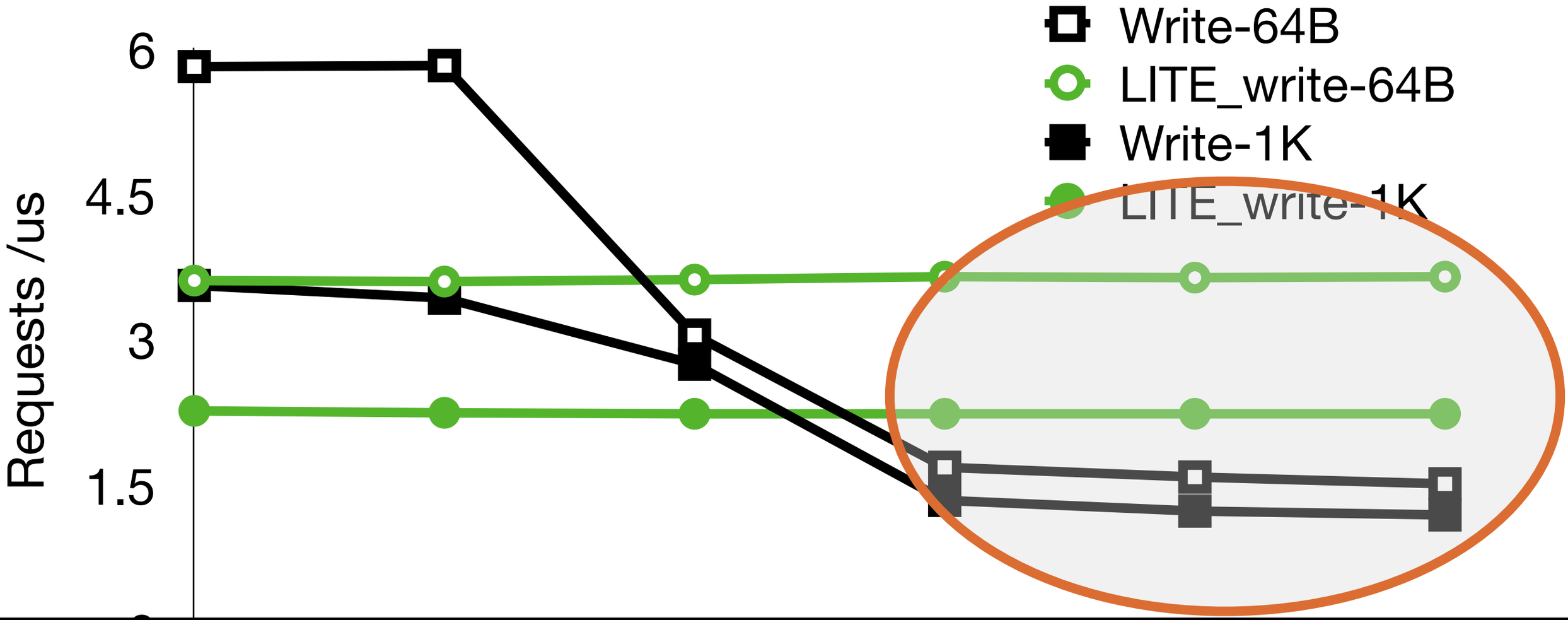
LITE RDMA:Size of MR Scalability



LITE RDMA:Size of MR Scalability

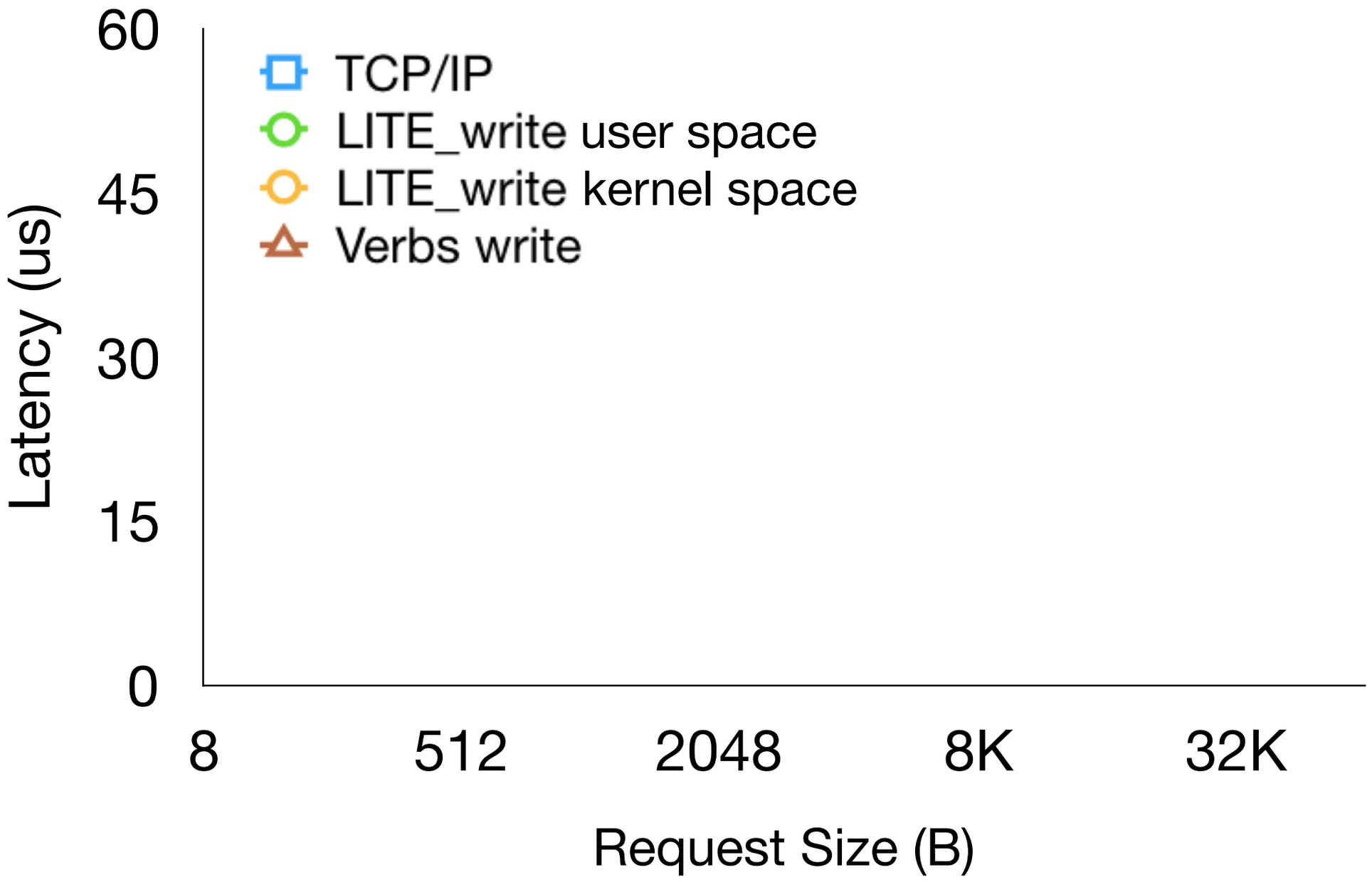


LITE RDMA:Size of MR Scalability

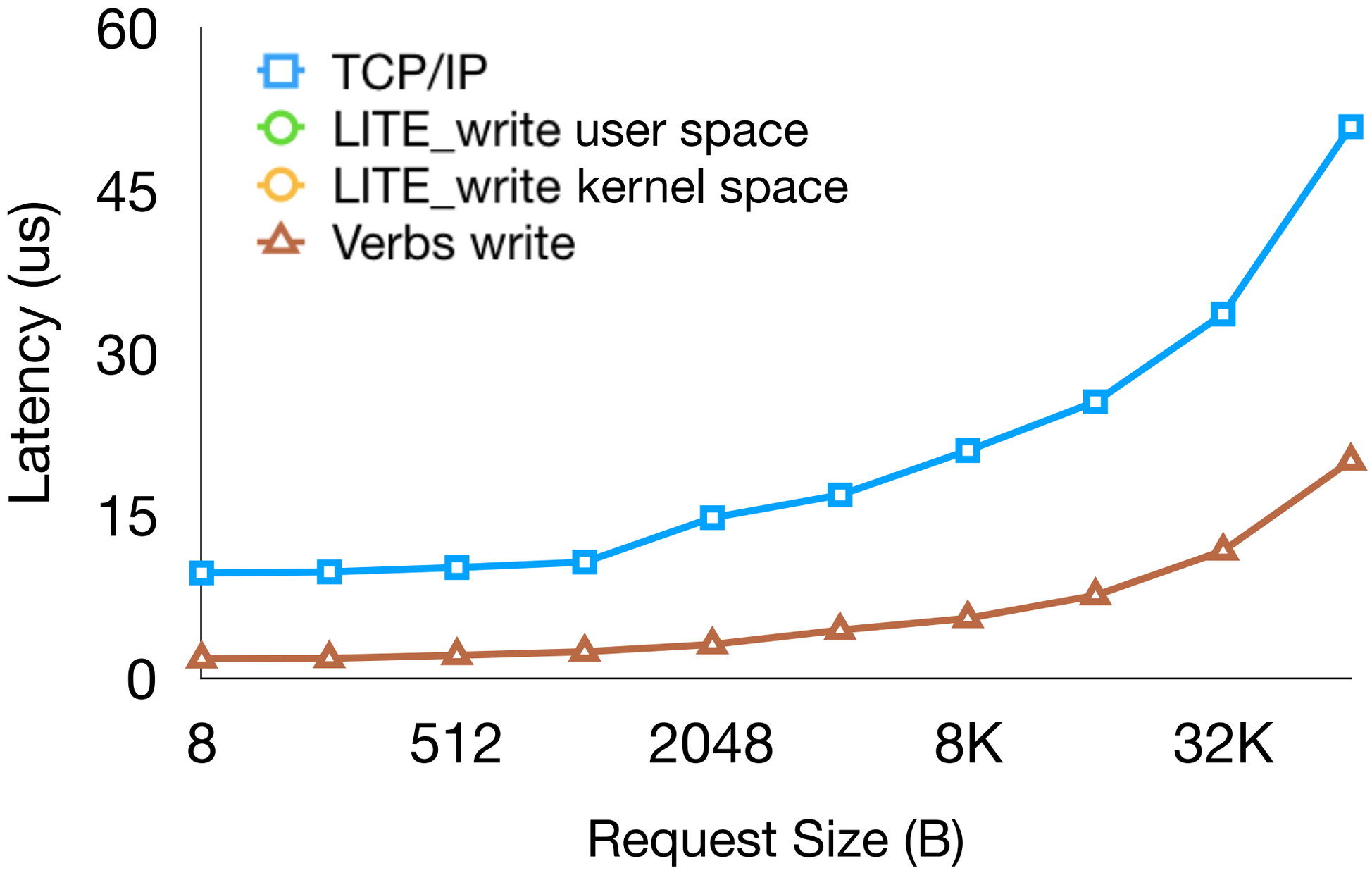


LITE scales much better than native RDMA wrt MR size and numbers

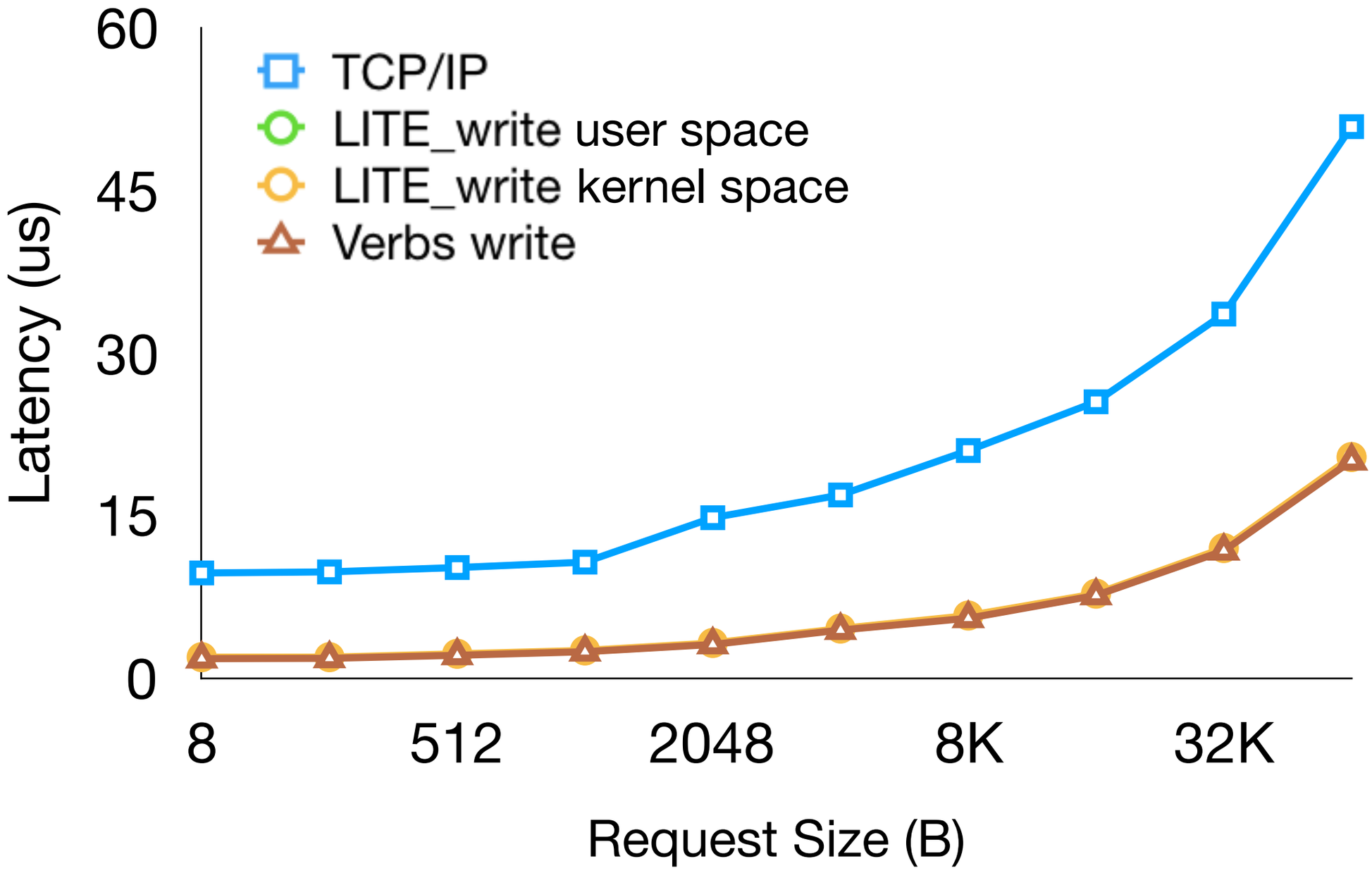
LITE-RDMA Latency



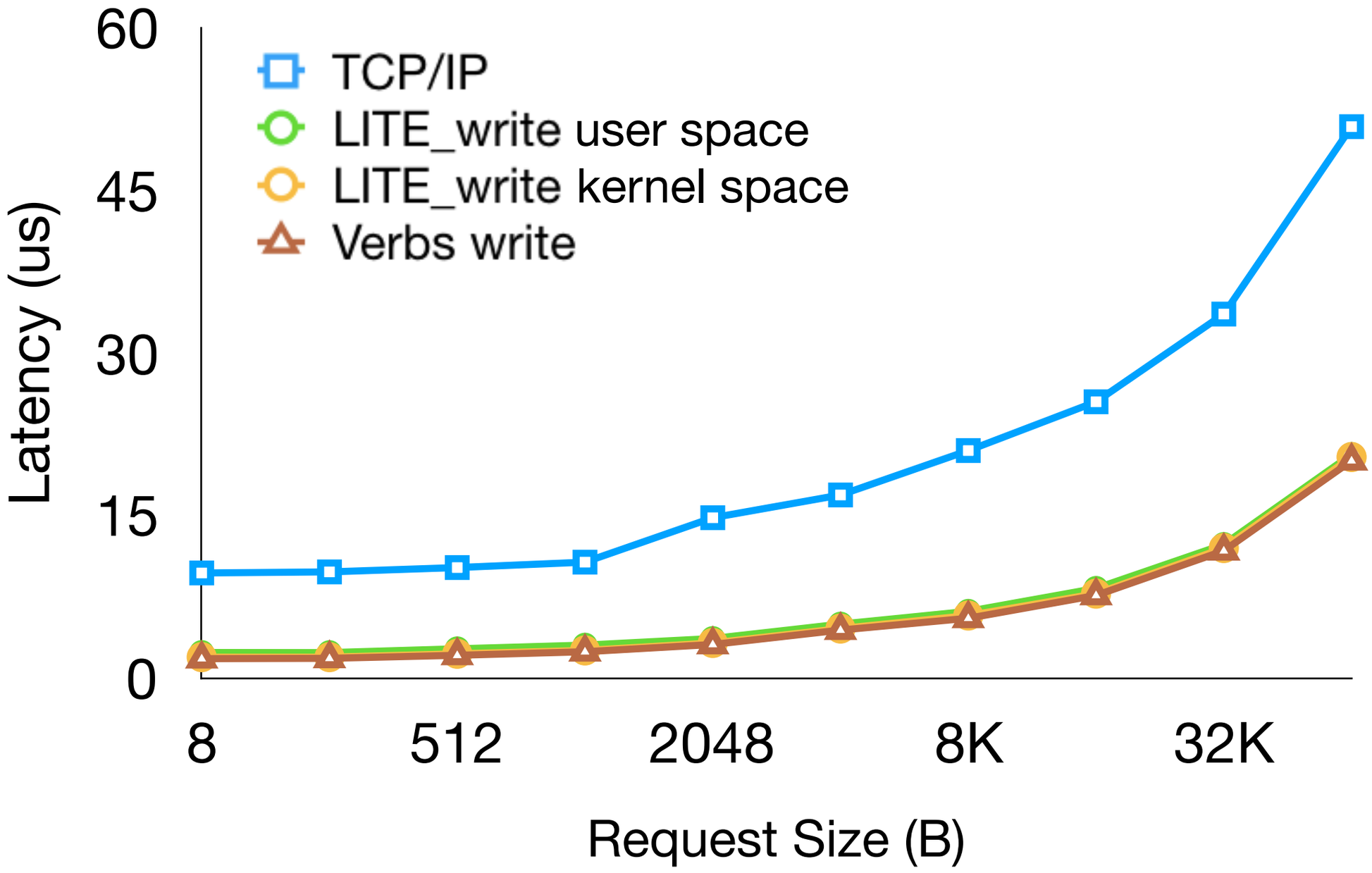
LITE-RDMA Latency



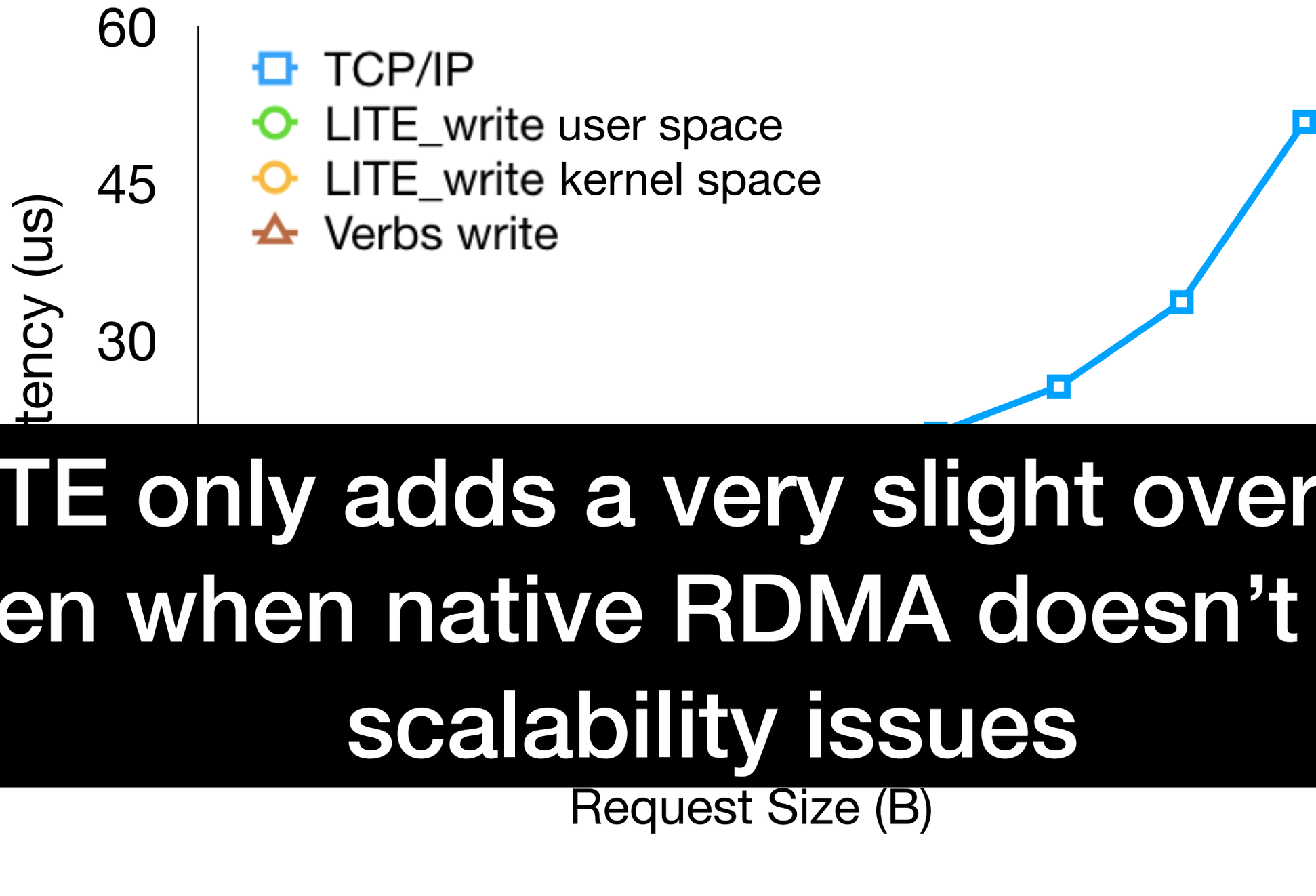
LITE-RDMA Latency



LITE-RDMA Latency



LITE-RDMA Latency



LITE only adds a very slight overhead even when native RDMA doesn't have scalability issues

LITE RPC

- RPC communication using two RDMA-write-imm
- One global busy poll thread
- Separate LMRs at server for different RPC clients
- Hide syscall cost behind performance critical path

- Benefits
  - Low latency
  - Low memory utilization
  - Low CPU utilization

Outline

- Introduction and motivation
- Overall design and abstraction
- LITE internals
- LITE applications
- Conclusion

LITE Application Effort

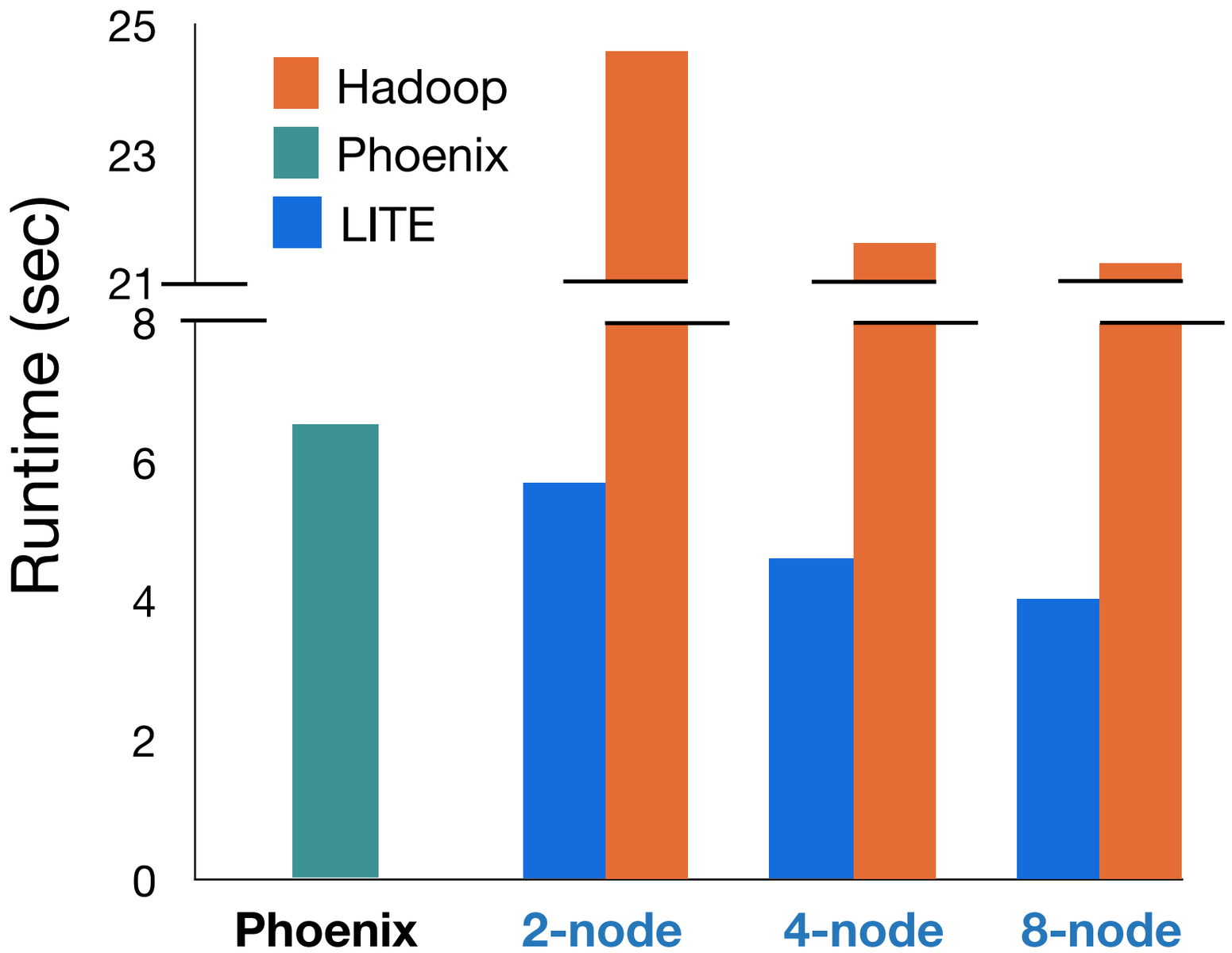
Application	LOC	LOC using LITE	Student Days
LITE-Log	330	36	1
LITE-MapReduce	600*	49	4
LITE-Graph	1400	20	7
LITE-Kernel-DSM	3000	45	26
LITE-Graph-DSM	1300	0	5

- Simple to use
- Needs no expert knowledge
- Flexible, powerful abstraction
- Easy to achieve optimized performance

* LITE-MapReduce ports from the 3000-LOC Phoenix with 600 lines of change or addition

MapReduce Results

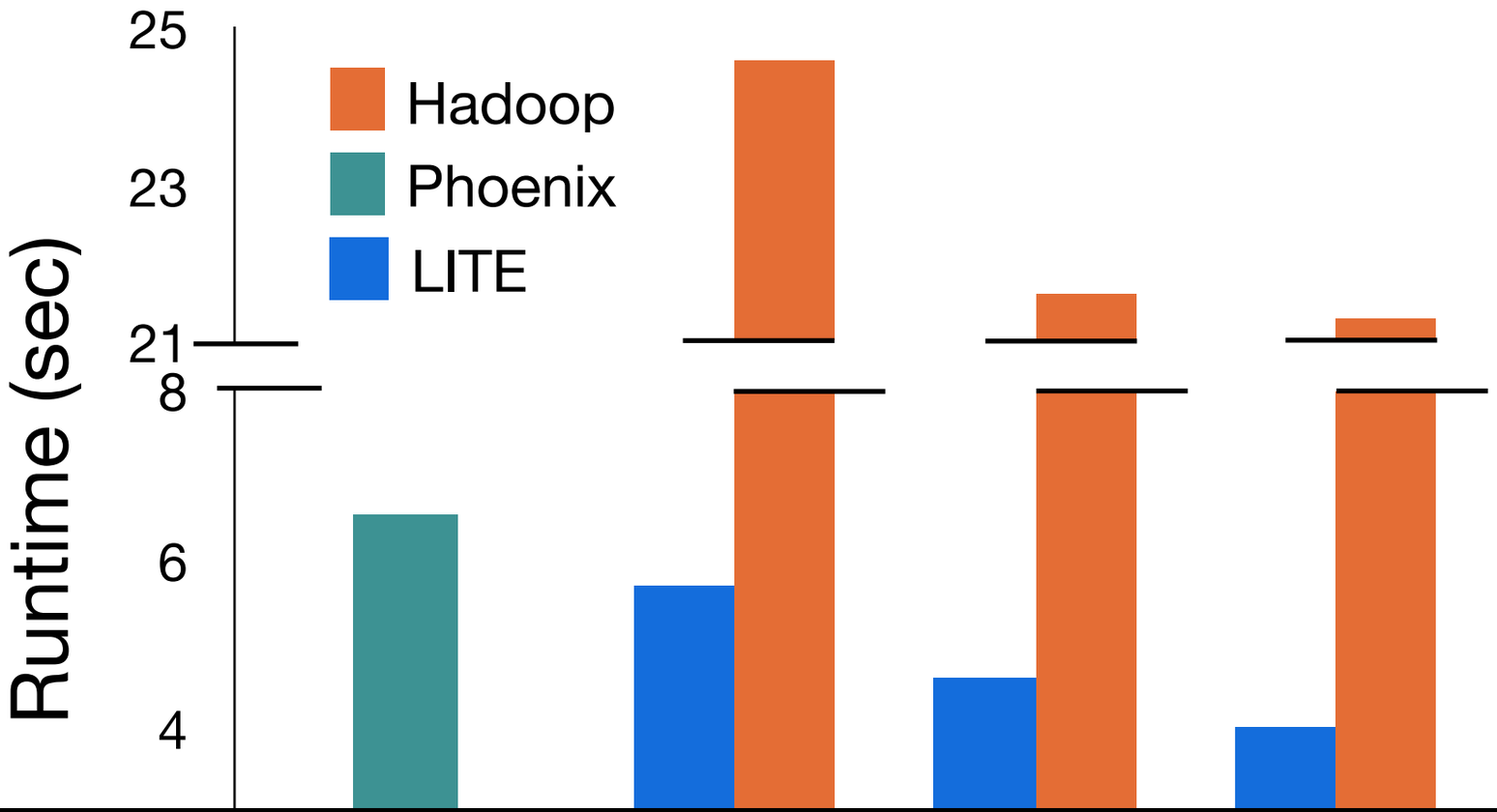
- LITE-MapReduce adapted from Phoenix [1]



[1]: “Ranger etal., Evaluating MapReduce for Multi-core and Multiprocessor Systems. (HPCA 07)”

MapReduce Results

- LITE-MapReduce adapted from Phoenix [1]

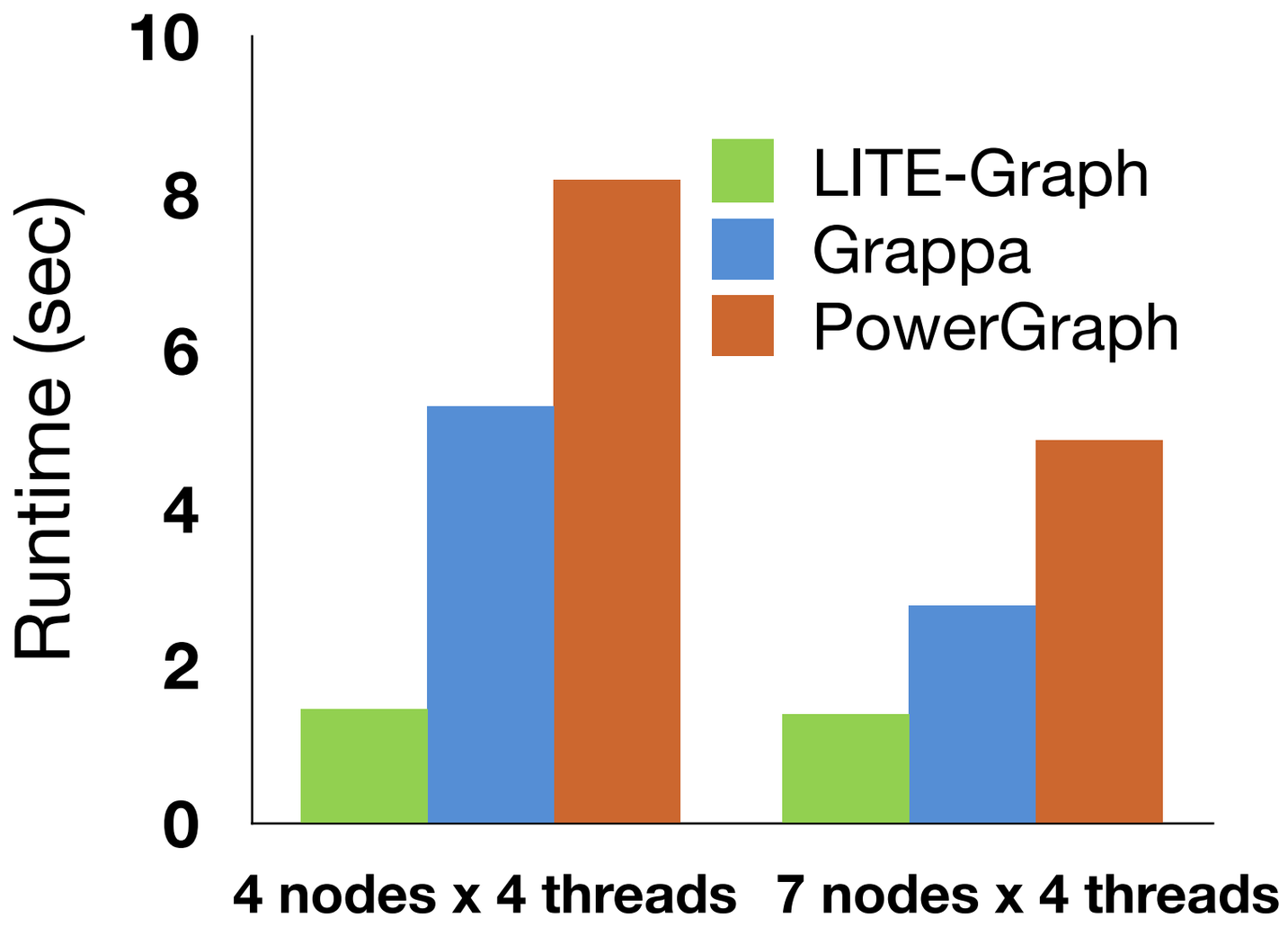


LITE-MapReduce outperforms Hadoop by 4.3x to 5.3x

[1]: "Ranger etal., Evaluating MapReduce for Multi-core and Multiprocessor Systems. (HPCA 07)"

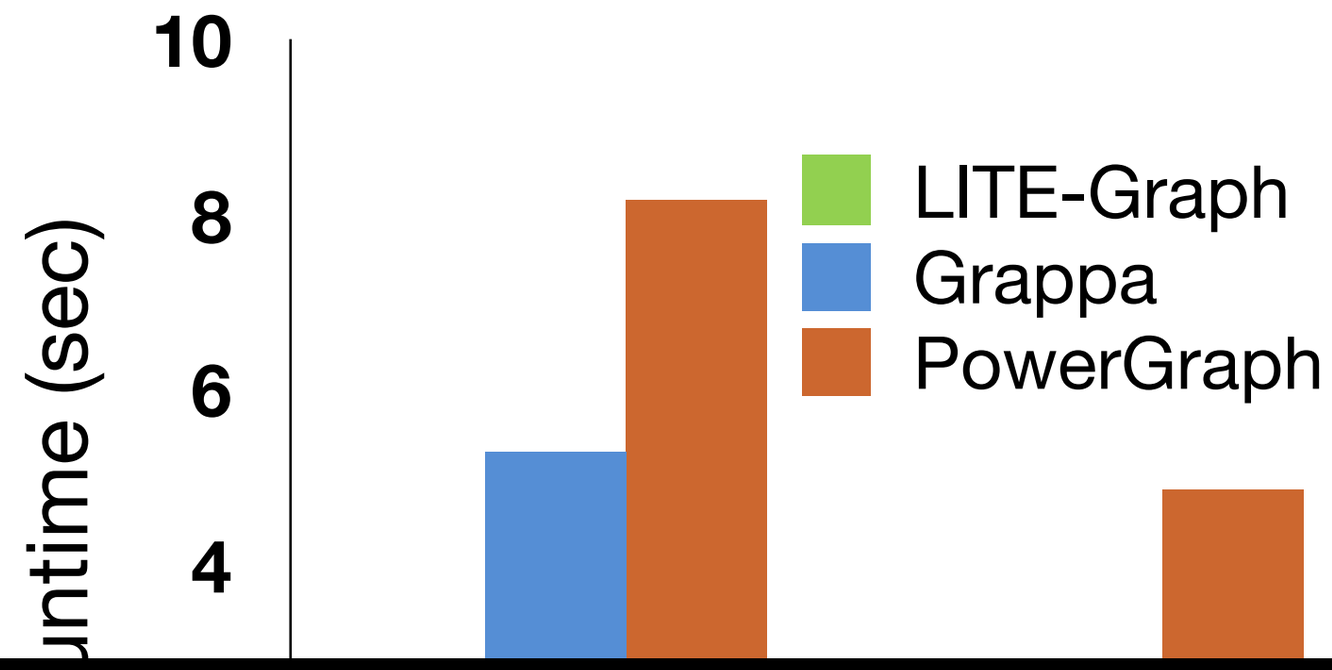
Graph Results

- LITE-Graph built directly on LITE using PowerGraph design
- Grappa and PowerGraph



Graph Results

- LITE-Graph built directly on LITE using PowerGraph design
- Grappa and PowerGraph



LITE-Graph outperforms PowerGraph by 3.5x to 5.6x

Conclusion

- LITE virtualizes RDMA into flexible abstraction
- LITE preserves RDMA's performance benefits
- ***Indirection*** not always degrade performance!

Conclusion

- LITE virtualizes RDMA into flexible abstraction
- LITE preserves RDMA's performance benefits
- ***Indirection*** not always degrade performance!
- Division across user space, kernel, and hardware

Thank you
Questions?

Get LITE at: *https://github.com/Wuklab/LITE*

WukLab

wuklab.io

PURDUE UNIVERSITY