

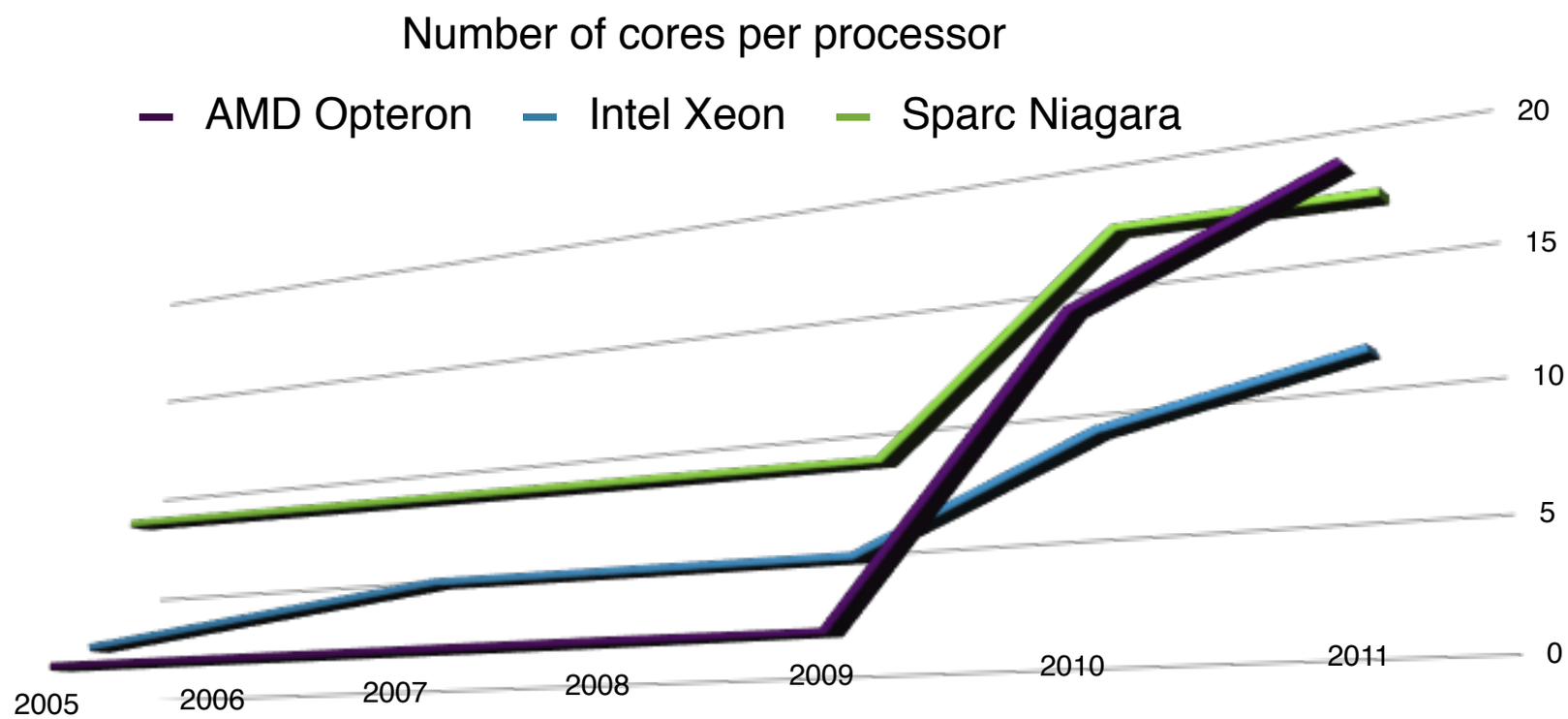


# TAKING A VIRTUAL MACHINE TOWARDS MANY-CORES

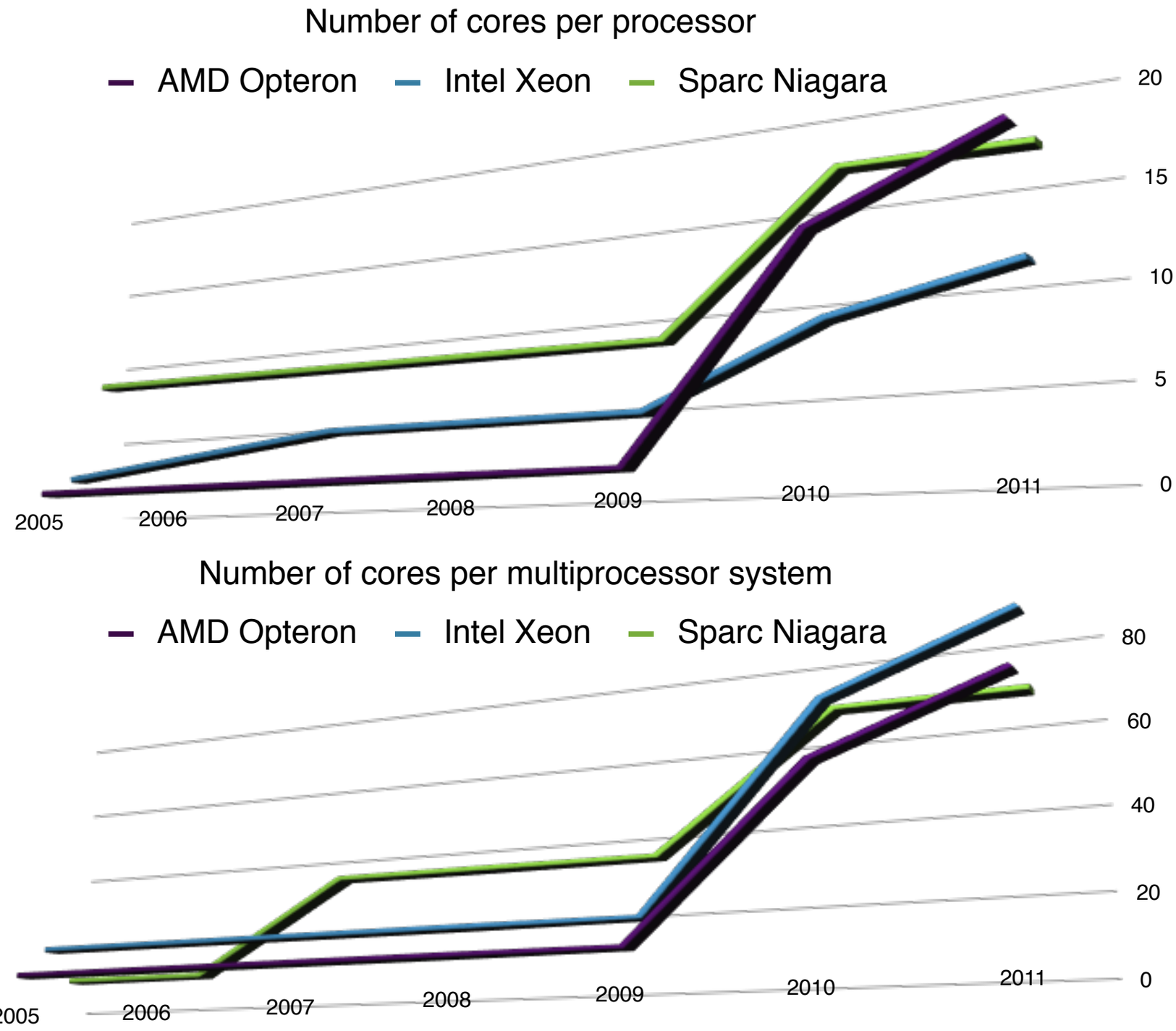
RICKARD GREEN - *rickard@erlang.org*

PATRIK NYBLOM - *pan@erlang.org*

# WHAT WE ALL KNOW BY NOW



# WHAT WE ALL KNOW BY NOW



# Resource Contention



# Resource Contention

- High level algorithms
  - Server
  - ...



# Resource Contention

- High level algorithms
  - Server
  - ...
- Software synchronization mechanisms
  - Locks
    - Lock type
    - Lock implementation
  - Lock free data structures
  - ...



ERLANG

# Resource Contention

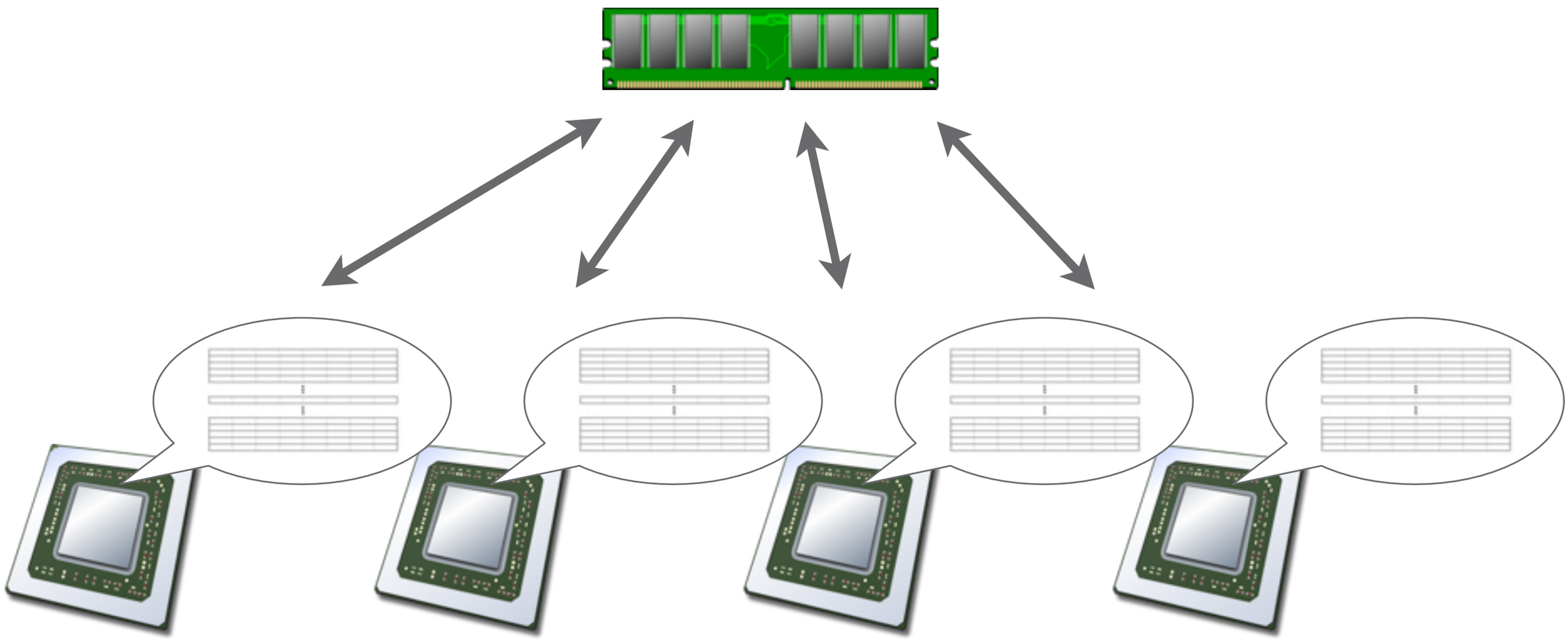
- High level algorithms
  - Server
  - ...
- Software synchronization mechanisms
  - Locks
    - Lock type
    - Lock implementation
  - Lock free data structures
  - ...
- Hardware
  - Processor communication
    - Cache line
    - Memory barrier
  - ...

# Resource Contention

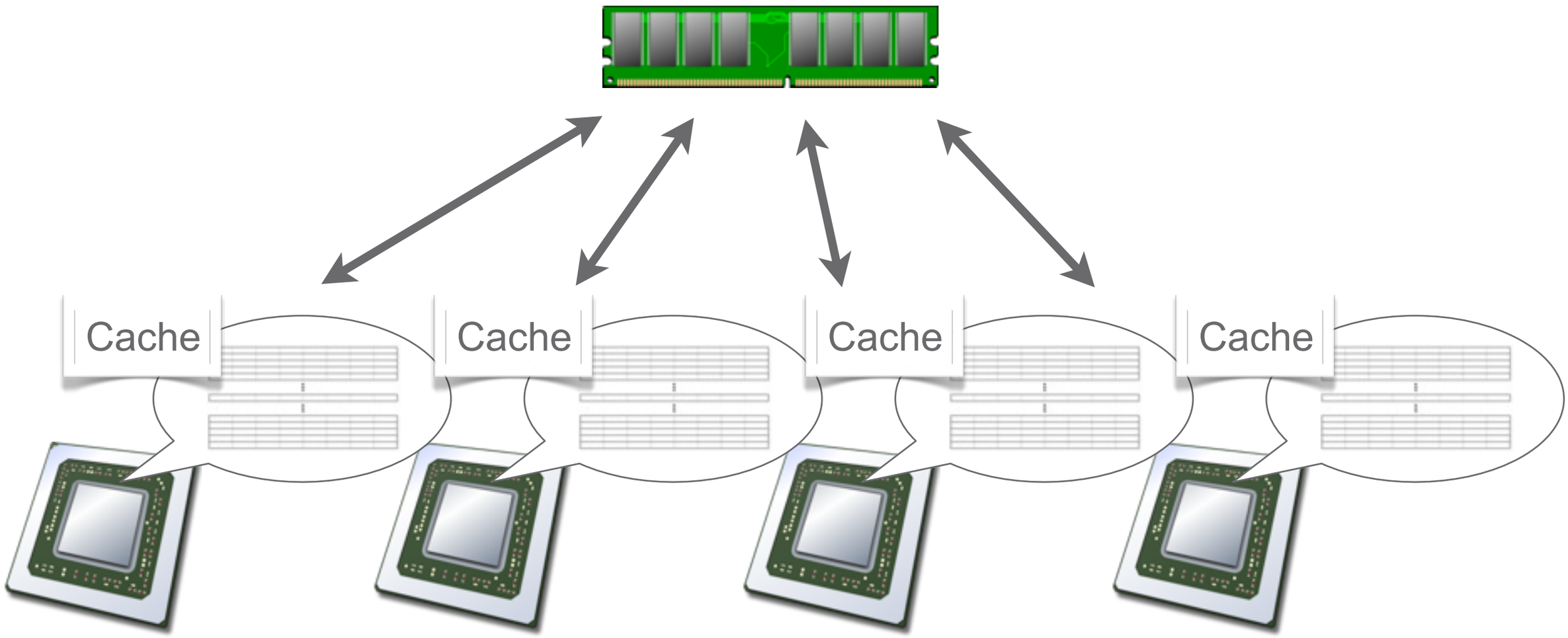
- High level algorithms
  - Server
  - ...
- Software synchronization mechanisms
  - Locks
    - Lock type
    - Lock implementation
  - Lock free data structures
  - ...
- Hardware
  - Processor communication
    - Cache line
    - Memory barrier
  - ...

Awareness

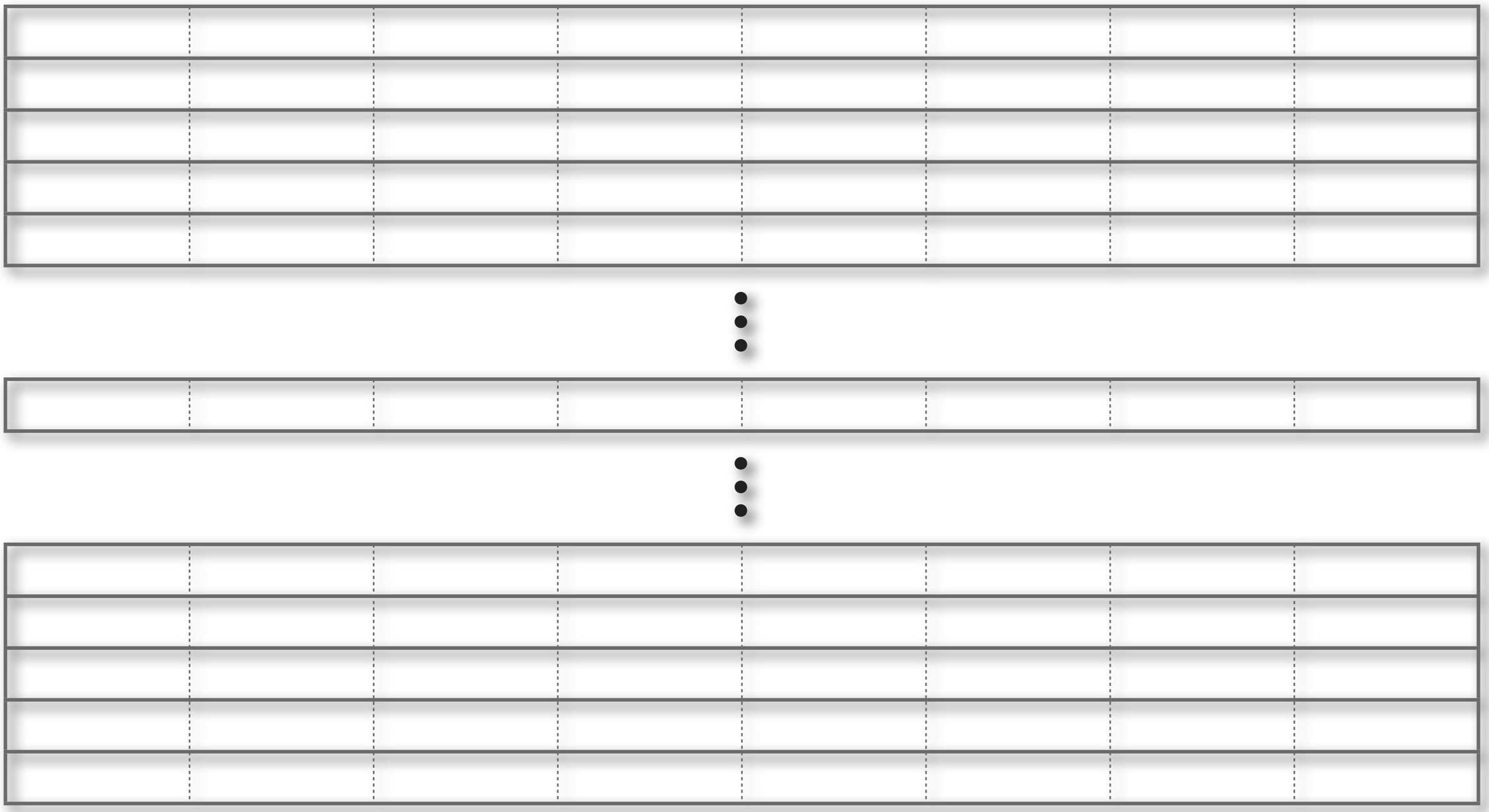
# SHARED MEMORY MULTIPROCESSOR SYSTEM



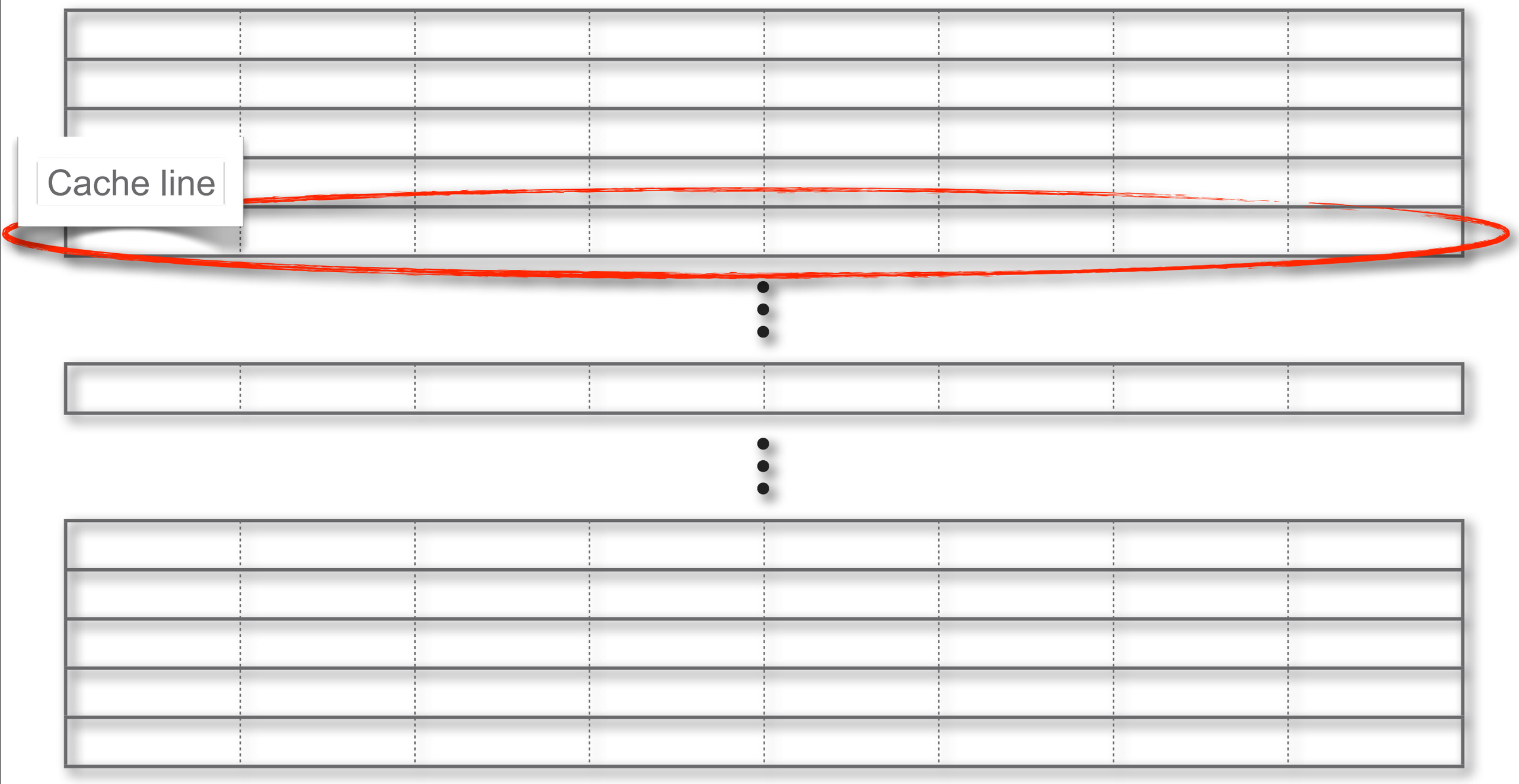
# SHARED MEMORY MULTIPROCESSOR SYSTEM



# CACHE



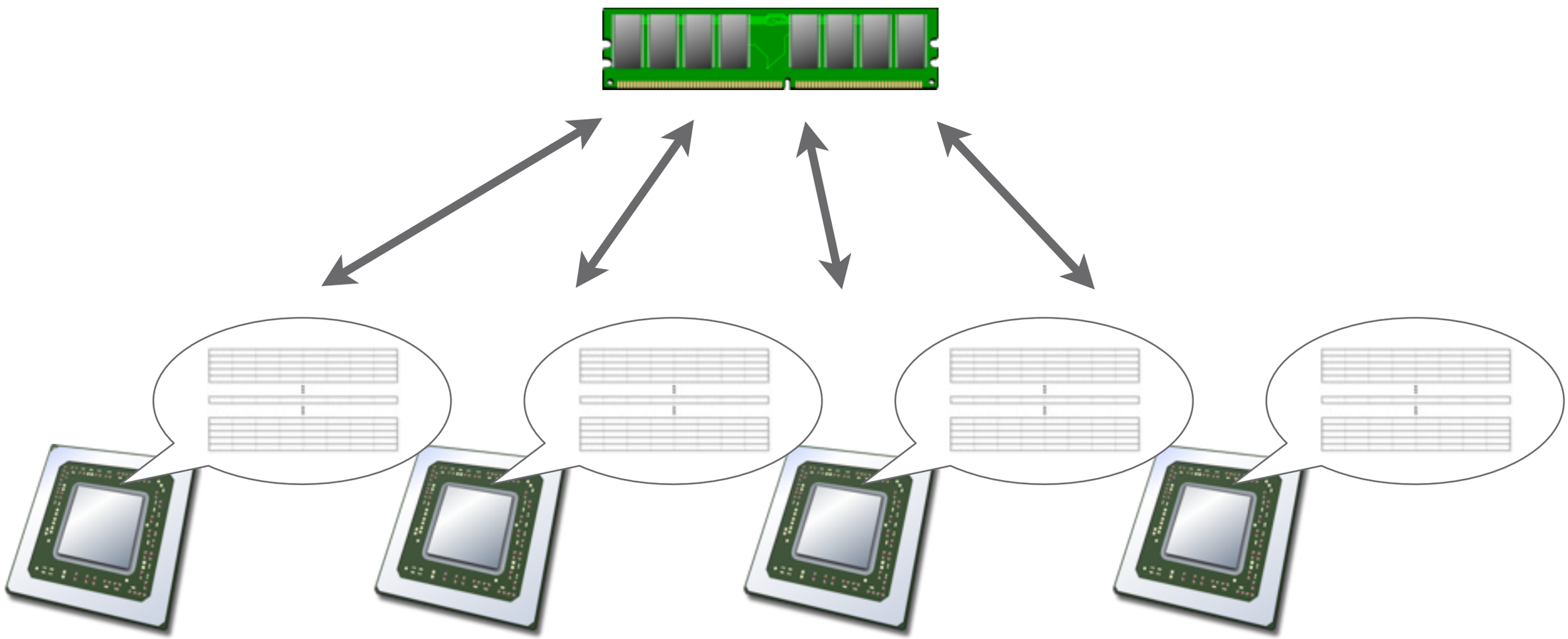
# CACHE



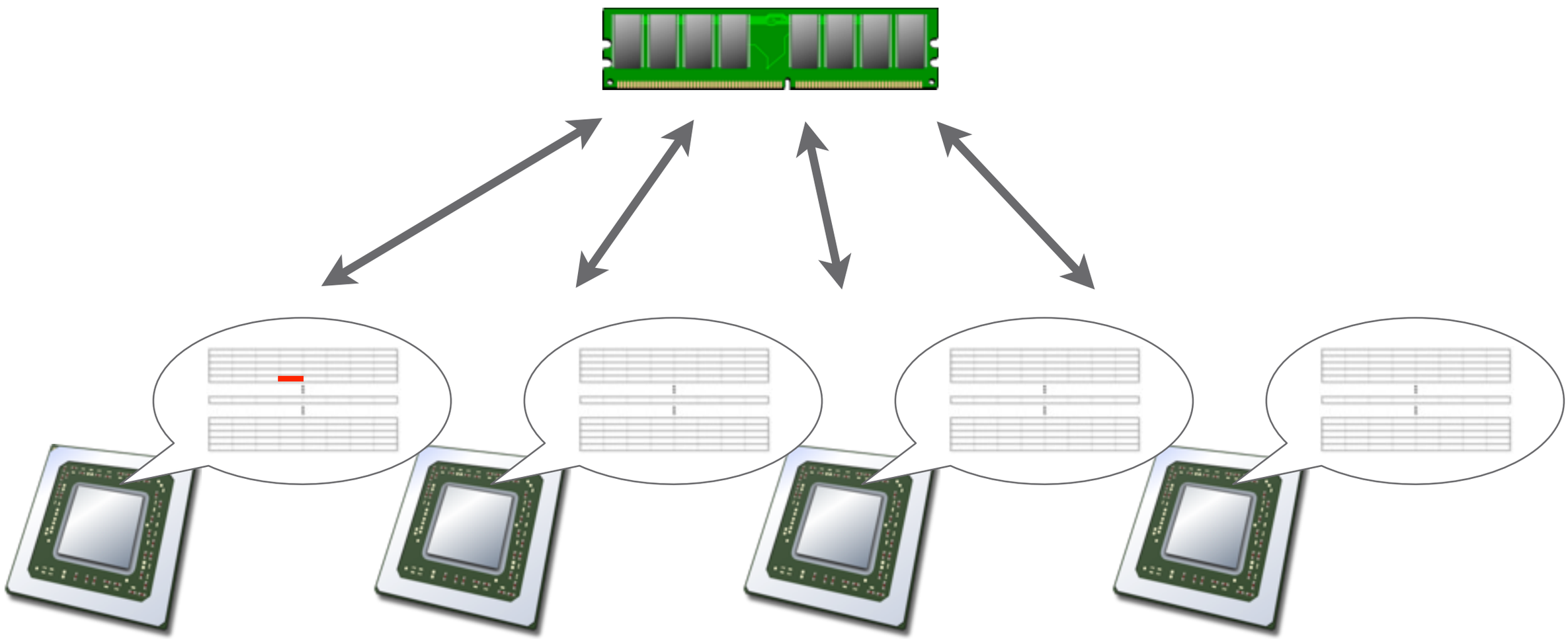



ERLANG

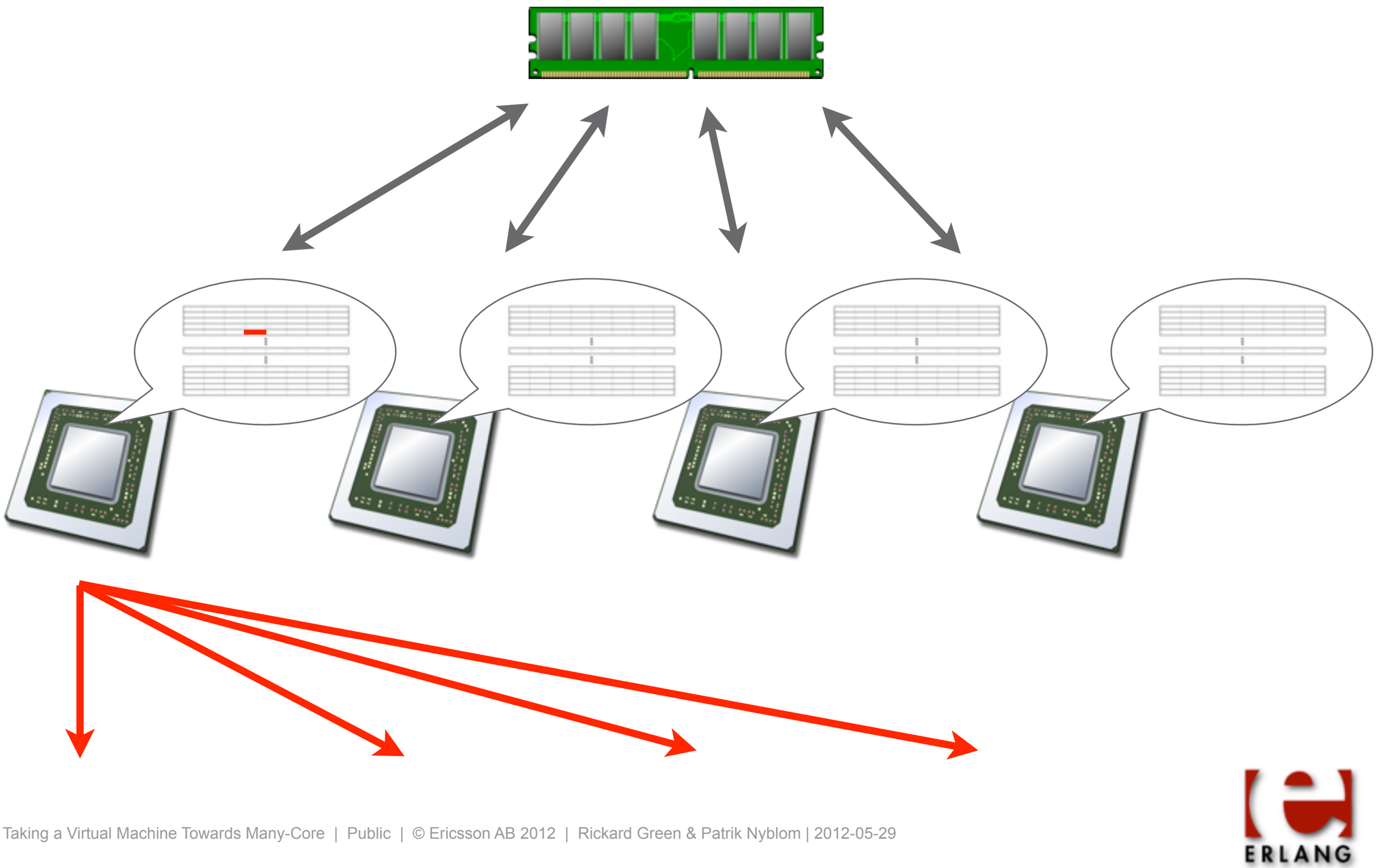
# SHARED MEMORY MULTIPROCESSOR SYSTEM



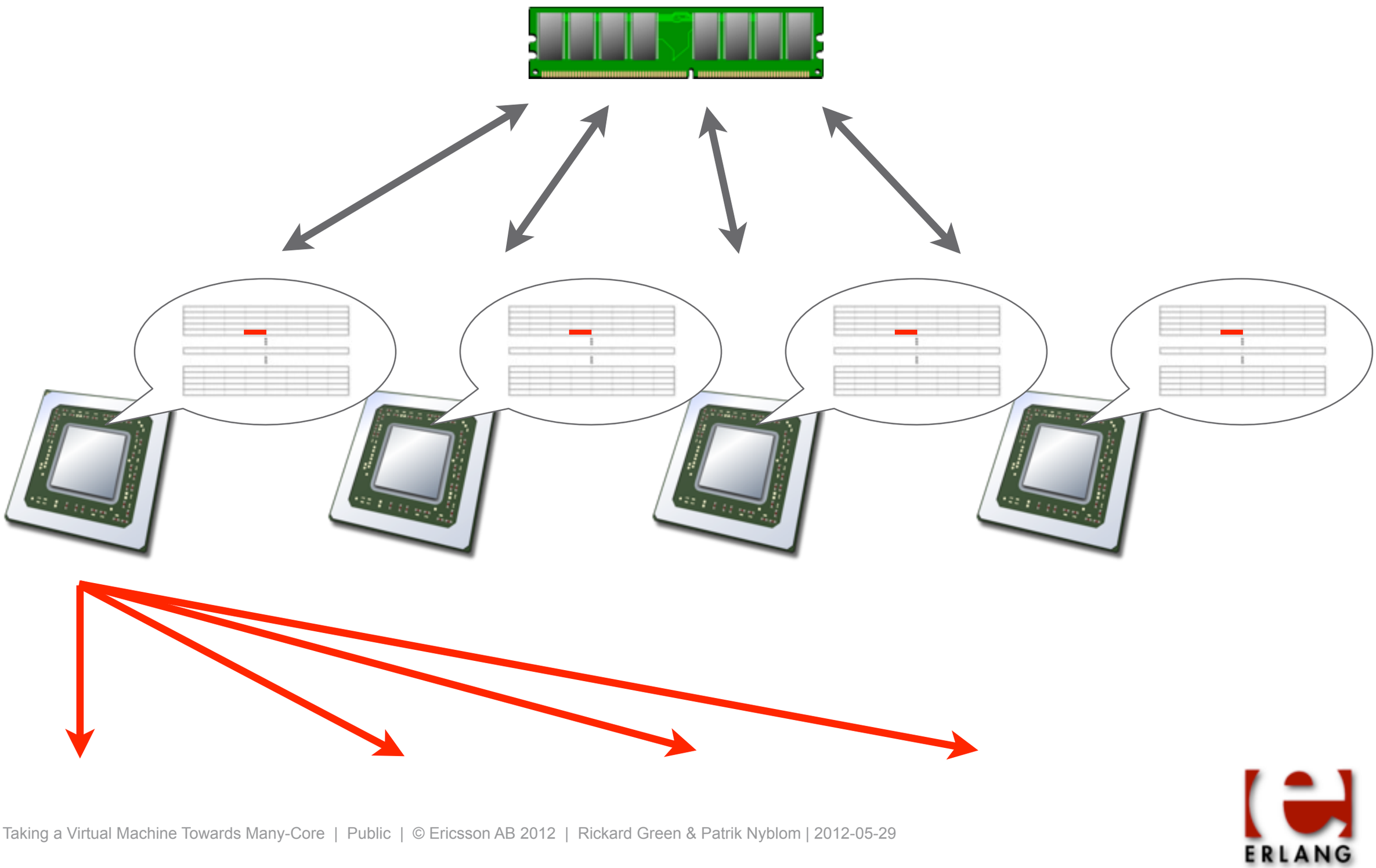
# SHARED MEMORY MULTIPROCESSOR SYSTEM



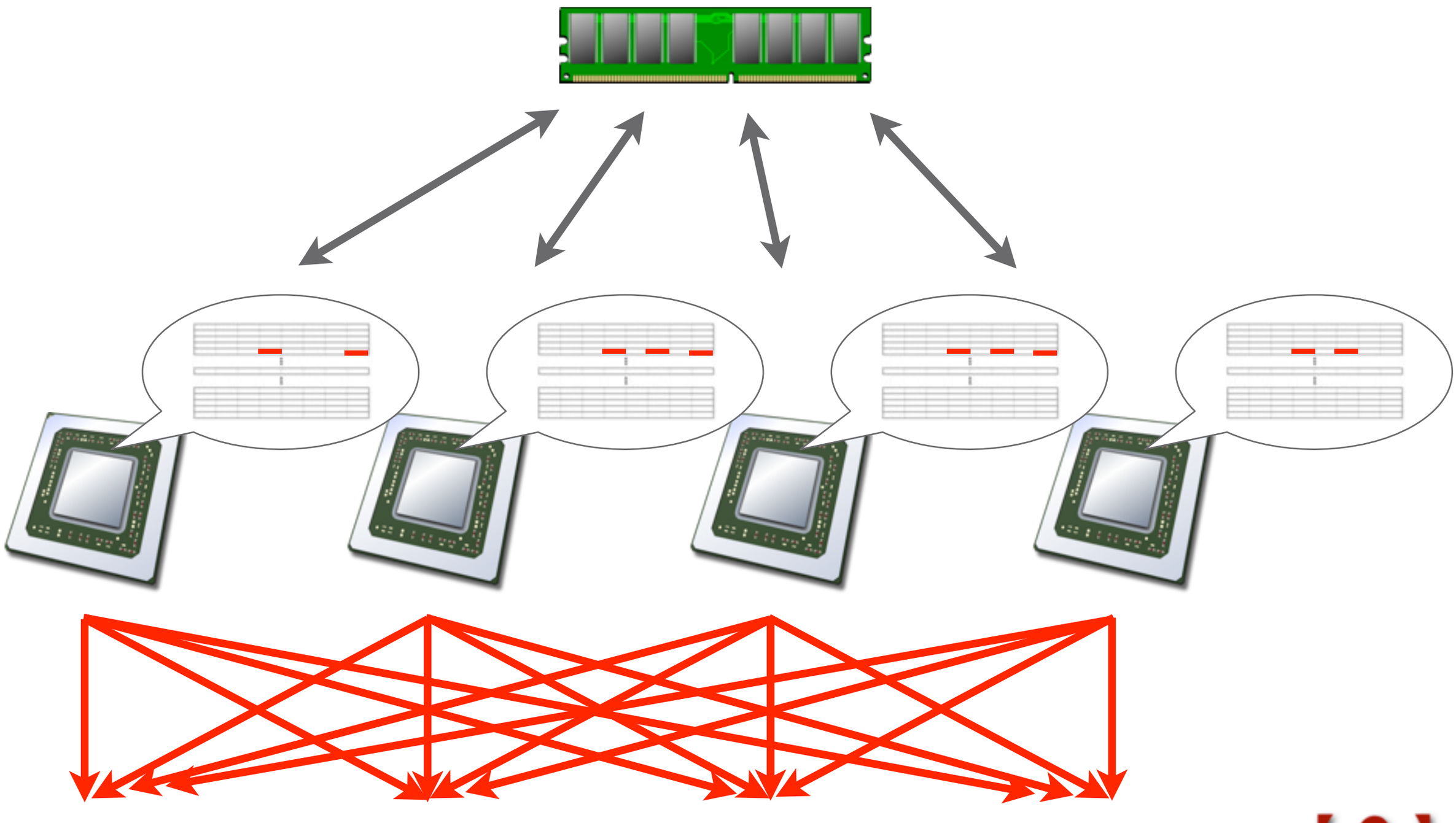
# SHARED MEMORY MULTIPROCESSOR SYSTEM



# SHARED MEMORY MULTIPROCESSOR SYSTEM



# SHARED MEMORY MULTIPROCESSOR SYSTEM



# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero

# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero

# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero



ERLANG

# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero



ERLANG

# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero

- Read-lock table
- Lookup element
- Read-unlock table
- Operate on element
- Deletion of element
  - Remove from table
  - Add number of schedulers to element reference count
  - Decrement reference count once for lost table reference
  - Schedule “confirm deletion jobs” on all schedulers
  - Each scheduler decrease reference count and release if it reached zero



ERLANG

# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero

- Read-lock table
- Lookup element
- Read-unlock table
- Operate on element
- Deletion of element
  - Remove from table
  - Add number of schedulers to element reference count
  - Decrement reference count once for lost table reference
  - Schedule “confirm deletion jobs” on all schedulers
  - Each scheduler decrease reference count and release if it reached zero

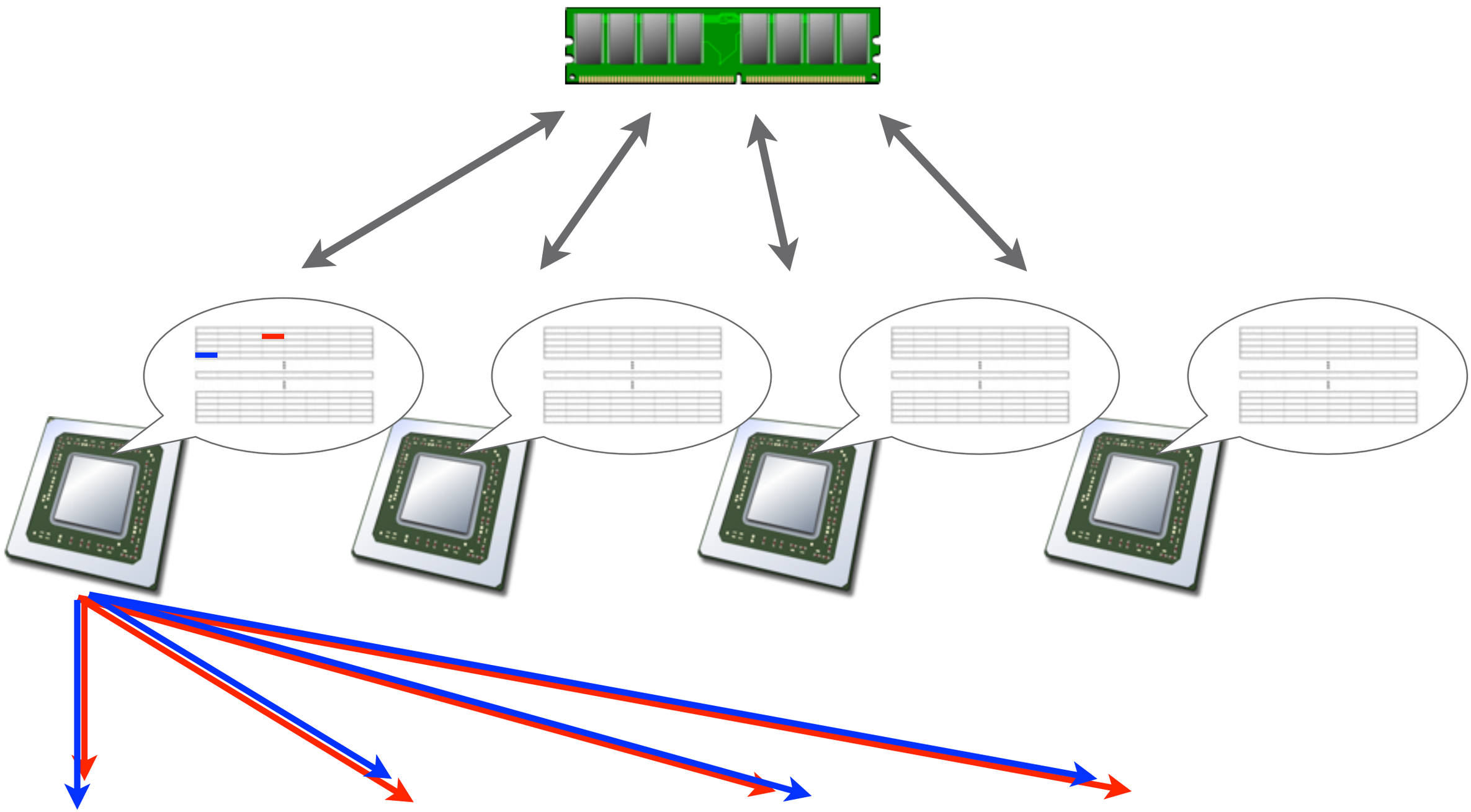
# TABLE ELEMENT LOOKUP

- Read-lock table
- Lookup element
- Increment element reference count
- Read-unlock table
- Operate on element
- Decrease element reference count
  - Release if reference count reached zero
- Deletion of element
  - Remove from table
  - Decrease reference count and release if it reached zero

- Lookup element
- Operate on element
- Deletion of element
  - Remove from table
  - Add number of schedulers to element reference count
  - Decrement reference count once for lost table reference
  - Schedule “confirm deletion jobs” on all schedulers
  - Each scheduler decrease reference count and release if it reached zero



# ORDER OF MEMORY OPERATIONS



# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|



# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|
| 0 | 0 |

# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|
| 0 | 0 |
| 0 | 1 |



# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |



# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

# ORDER OF MEMORY OPERATIONS

c1 = c2 = 1

| Thread 1 | Thread 2 |
|---|---|
| Store(c1, 0) | Store(c2, 0) |
| MemoryBarrier | MemoryBarrier |
| r1 = Load(c2) | r2 = Load(c1) |

| r1 | r2 |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |

# HARDWARE MEMORY BARRIERS

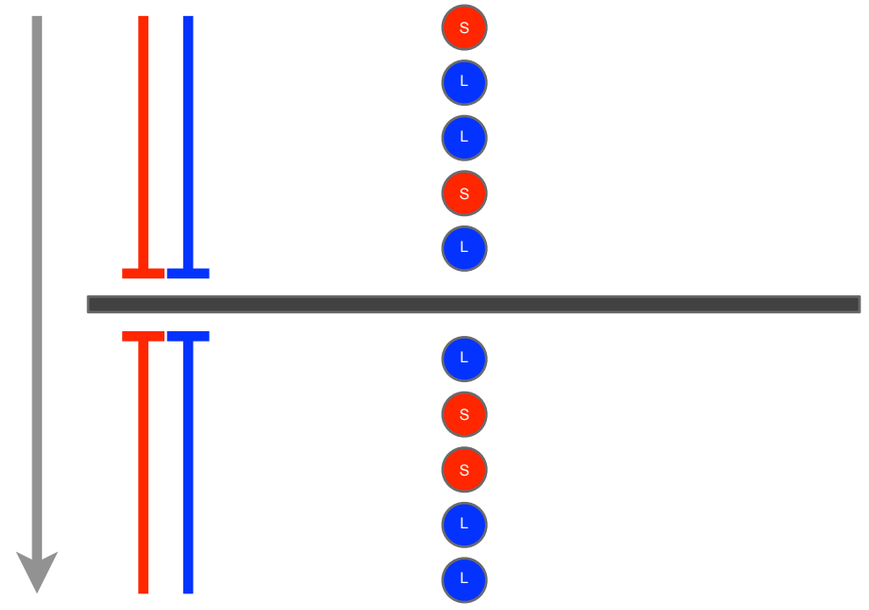
- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist



ERLANG

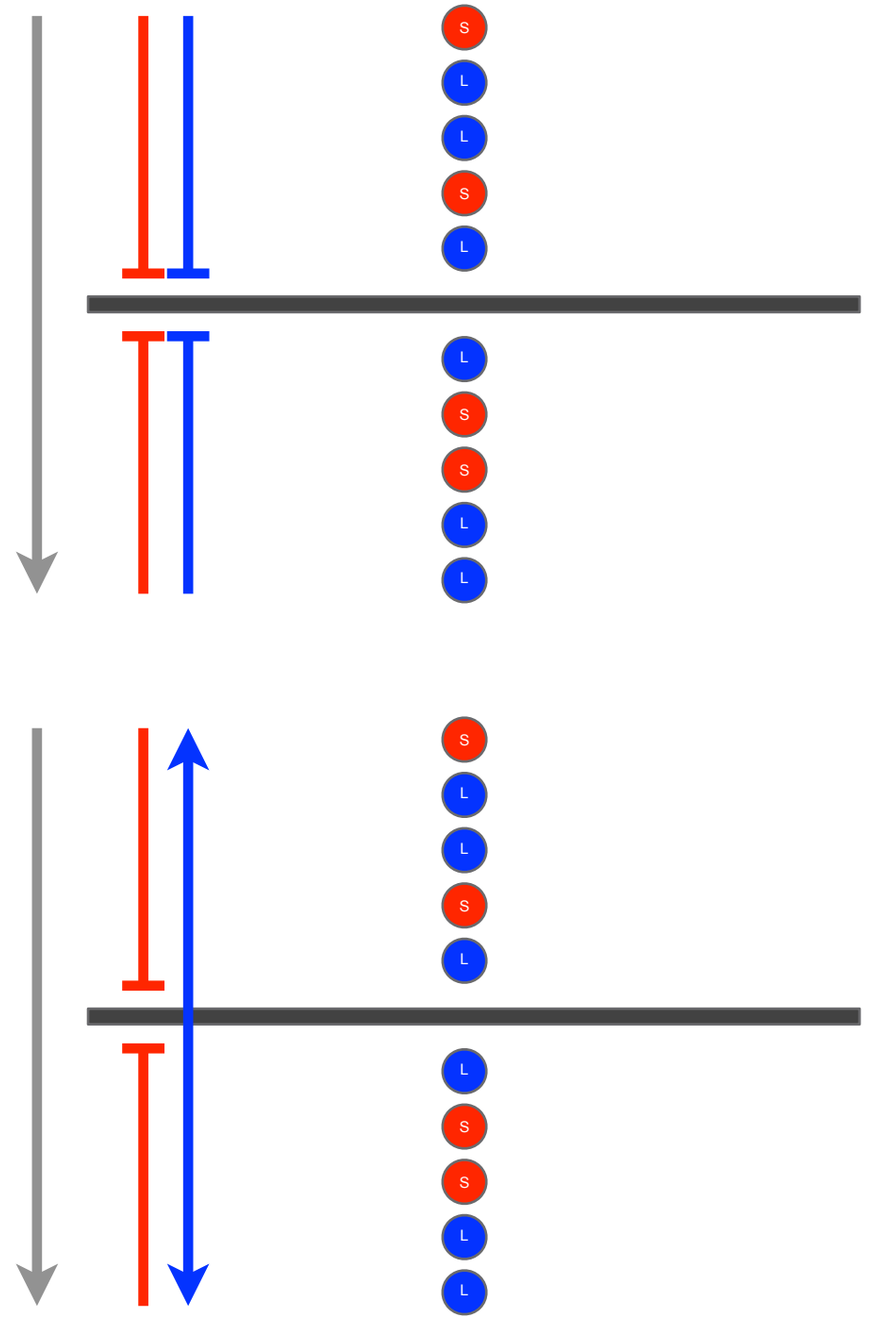
# HARDWARE MEMORY BARRIERS

- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist



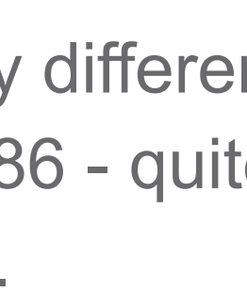
# HARDWARE MEMORY BARRIERS

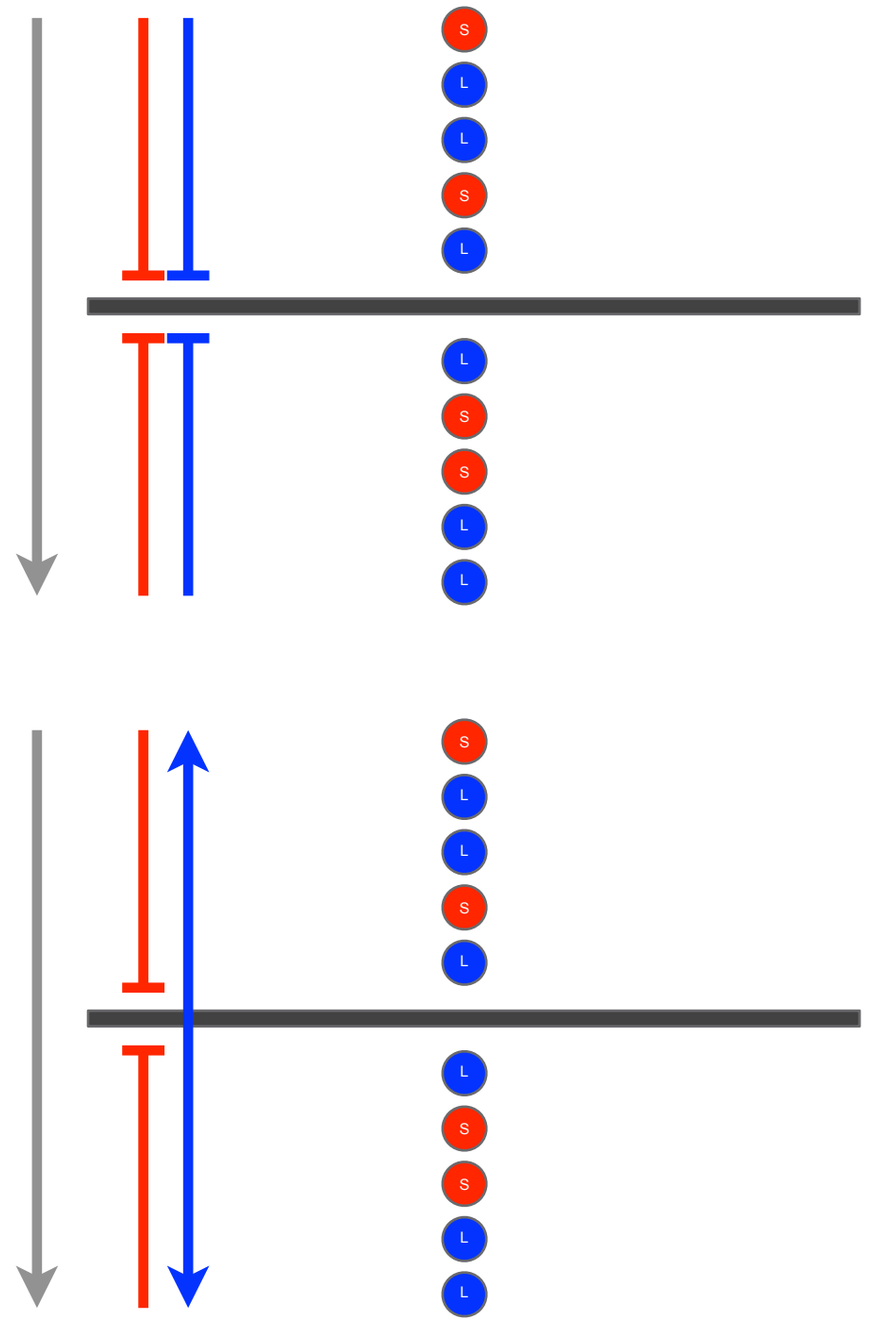
- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist



# HARDWARE MEMORY BARRIERS

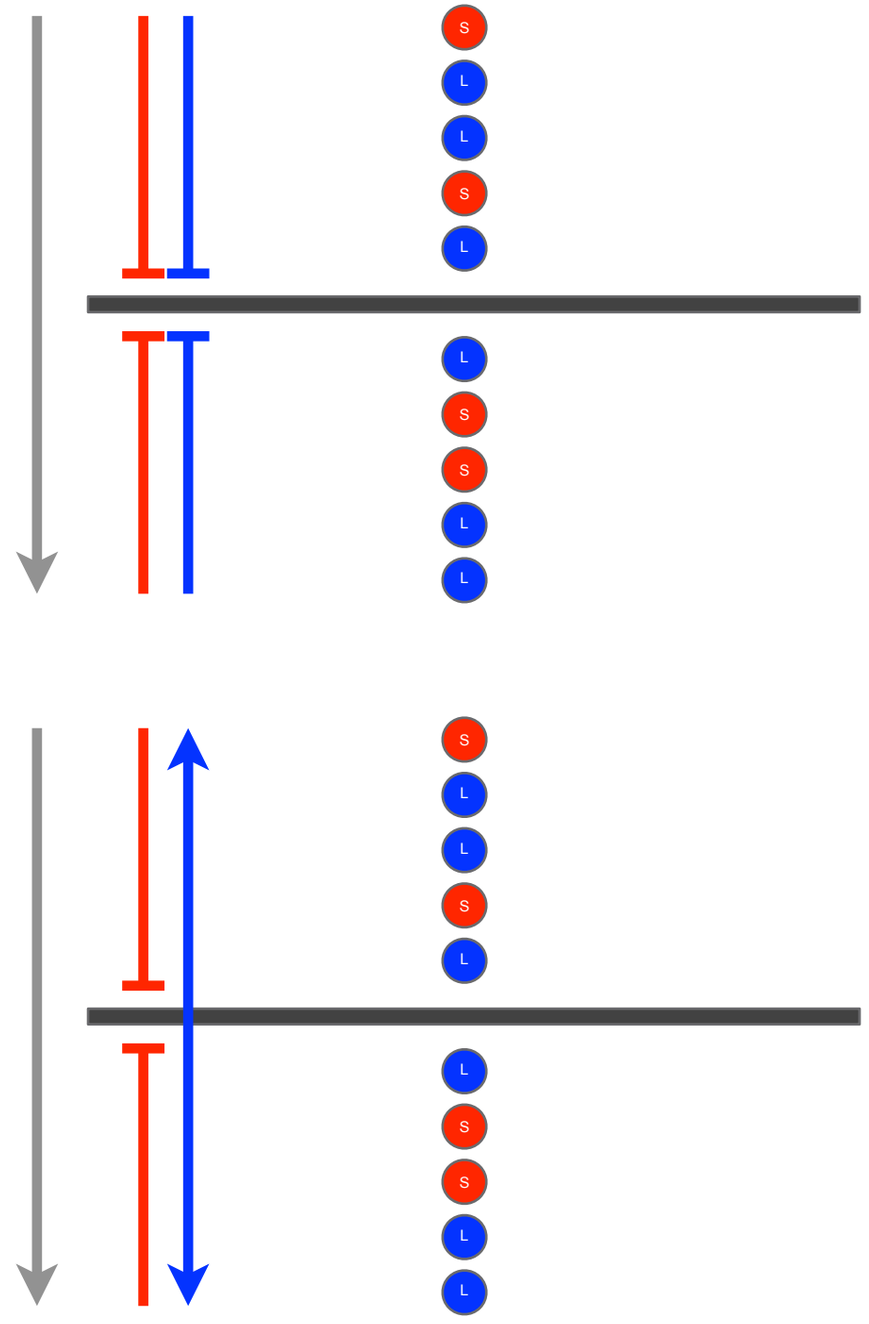
- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist

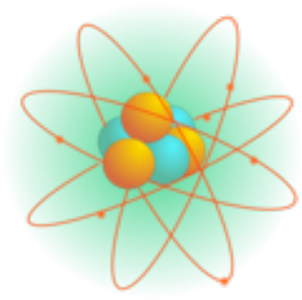




# HARDWARE MEMORY BARRIERS

- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist



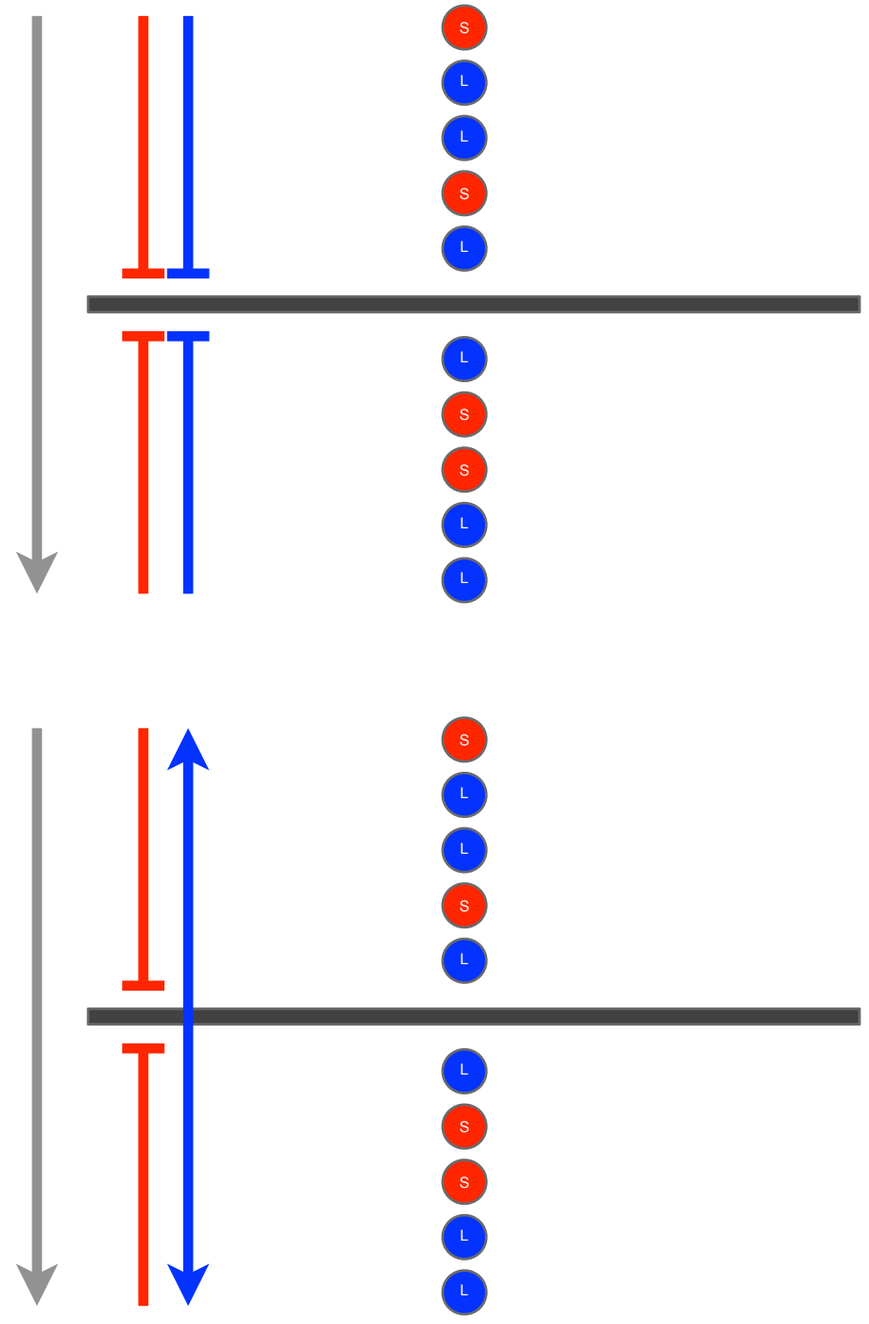



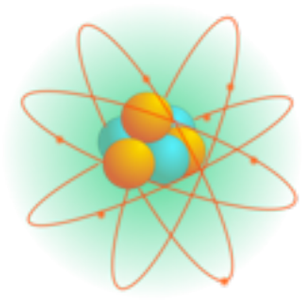


ERLANG

# HARDWARE MEMORY BARRIERS

- Hardware architectures
  - Very different ordering guarantees
    - x86 - quite strict
    - ...
    - alpha - very relaxed
  - Hardware memory barriers
    - Tools for enforcing ordering
    - Expensive
    - Architecture dependent
    - Sometimes lightweight versions exist






ERLANG

# OLD ATOMIC API

ethr_atomic_init()
ethr_atomic_set()
ethr_atomic_read()
ethr_atomic_inc()
ethr_atomic_dec()
ethr_atomic_add()
ethr_atomic_inc_read_mb()
ethr_atomic_dec_read_mb()
ethr_atomic_add_read_mb()
ethr_atomic_read_bor_mb()
ethr_atomic_read_band_mb()
ethr_atomic_xchg_mb()
ethr_atomic_cmpxchg_mb()

# OLD ATOMIC API

## Word size

ethr_atomic_init()
ethr_atomic_set()
ethr_atomic_read()
ethr_atomic_inc()
ethr_atomic_dec()
ethr_atomic_add()
ethr_atomic_inc_read_mb()
ethr_atomic_dec_read_mb()
ethr_atomic_add_read_mb()
ethr_atomic_read_bor_mb()
ethr_atomic_read_band_mb()
ethr_atomic_xchg_mb()
ethr_atomic_cmpxchg_mb()



# ATOMIC API R15



# ATOMIC API R15

## 32-bit

ethr_atomic32_init()
ethr_atomic32_set()
ethr_atomic32_read()
ethr_atomic32_inc_read()
ethr_atomic32_dec_read()
ethr_atomic32_inc()
ethr_atomic32_dec()
ethr_atomic32_add_read()
ethr_atomic32_add()
ethr_atomic32_read_bor()
ethr_atomic32_read_band()
ethr_atomic32_xchg()
ethr_atomic32_cmpxchg()
ethr_atomic32_init_mb()
ethr_atomic32_set_mb()
ethr_atomic32_read_mb()
ethr_atomic32_inc_read_mb()
ethr_atomic32_dec_read_mb()
ethr_atomic32_inc_mb()
ethr_atomic32_dec_mb()
ethr_atomic32_add_read_mb()
ethr_atomic32_add_mb()
ethr_atomic32_read_bor_mb()
ethr_atomic32_read_band_mb()
ethr_atomic32_xchg_mb()
ethr_atomic32_cmpxchg_mb()
ethr_atomic32_init_acqb()
ethr_atomic32_set_acqb()
ethr_atomic32_read_acqb()
ethr_atomic32_inc_read_acqb()
ethr_atomic32_dec_read_acqb()
ethr_atomic32_inc_acqb()
ethr_atomic32_dec_acqb()
ethr_atomic32_add_read_acqb()
ethr_atomic32_add_acqb()
ethr_atomic32_read_bor_acqb()
ethr_atomic32_read_band_acqb()
ethr_atomic32_xchg_acqb()
ethr_atomic32_cmpxchg_acqb()
ethr_atomic32_init_relb()
ethr_atomic32_set_relb()
ethr_atomic32_read_relb()
ethr_atomic32_inc_read_relb()
ethr_atomic32_dec_read_relb()
ethr_atomic32_inc_relb()
ethr_atomic32_dec_relb()
ethr_atomic32_add_read_relb()
ethr_atomic32_add_relb()
ethr_atomic32_read_bor_relb()
ethr_atomic32_read_band_relb()
ethr_atomic32_xchg_relb()
ethr_atomic32_cmpxchg_relb()
ethr_atomic32_init_ddrb()
ethr_atomic32_set_ddrb()
ethr_atomic32_read_ddrb()
ethr_atomic32_inc_read_ddrb()
ethr_atomic32_dec_read_ddrb()
ethr_atomic32_inc_ddrb()
ethr_atomic32_dec_ddrb()
ethr_atomic32_add_read_ddrb()
ethr_atomic32_add_ddrb()
ethr_atomic32_read_bor_ddrb()
ethr_atomic32_read_band_ddrb()
ethr_atomic32_xchg_ddrb()
ethr_atomic32_cmpxchg_ddrb()
ethr_atomic32_init_rb()
ethr_atomic32_set_rb()
ethr_atomic32_read_rb()
ethr_atomic32_inc_read_rb()
ethr_atomic32_dec_read_rb()
ethr_atomic32_inc_rb()
ethr_atomic32_dec_rb()
ethr_atomic32_add_read_rb()
ethr_atomic32_add_rb()
ethr_atomic32_read_bor_rb()
ethr_atomic32_read_band_rb()
ethr_atomic32_xchg_rb()
ethr_atomic32_cmpxchg_rb()
ethr_atomic32_init_wb()
ethr_atomic32_set_wb()
ethr_atomic32_read_wb()
ethr_atomic32_inc_read_wb()
ethr_atomic32_dec_read_wb()
ethr_atomic32_inc_wb()
ethr_atomic32_dec_wb()
ethr_atomic32_add_read_wb()
ethr_atomic32_add_wb()
ethr_atomic32_read_bor_wb()
ethr_atomic32_read_band_wb()
ethr_atomic32_xchg_wb()
ethr_atomic32_cmpxchg_wb()

## Word size

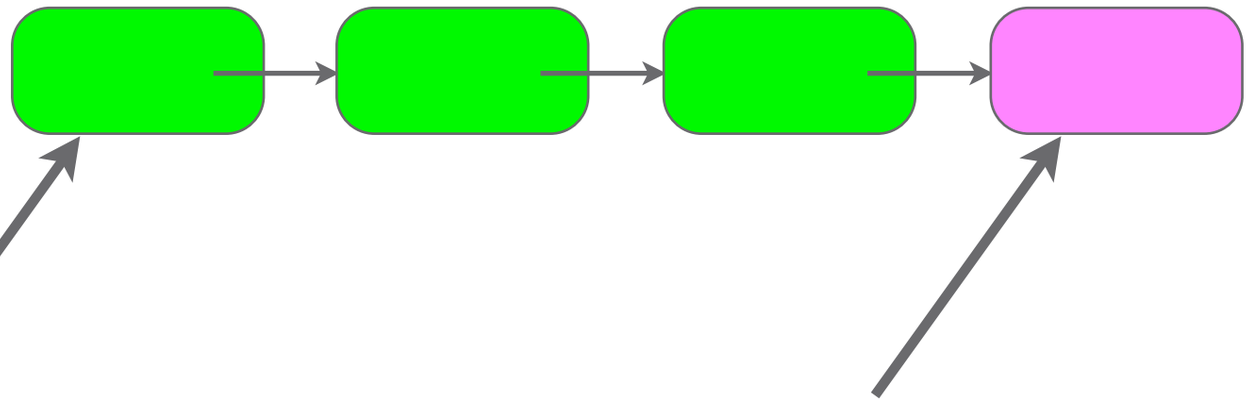
ethr_atomic_init()
ethr_atomic_set()
ethr_atomic_read()
ethr_atomic_inc_read()
ethr_atomic_dec_read()
ethr_atomic_inc()
ethr_atomic_dec()
ethr_atomic_add_read()
ethr_atomic_add()
ethr_atomic_read_bor()
ethr_atomic_read_band()
ethr_atomic_xchg()
ethr_atomic_cmpxchg()
ethr_atomic_init_mb()
ethr_atomic_set_mb()
ethr_atomic_read_mb()
ethr_atomic_inc_read_mb()
ethr_atomic_dec_read_mb()
ethr_atomic_inc_mb()
ethr_atomic_dec_mb()
ethr_atomic_add_read_mb()
ethr_atomic_add_mb()
ethr_atomic_read_bor_mb()
ethr_atomic_read_band_mb()
ethr_atomic_xchg_mb()
ethr_atomic_cmpxchg_mb()
ethr_atomic_init_acqb()
ethr_atomic_set_acqb()
ethr_atomic_read_acqb()
ethr_atomic_inc_read_acqb()
ethr_atomic_dec_read_acqb()
ethr_atomic_inc_acqb()
ethr_atomic_dec_acqb()
ethr_atomic_add_read_acqb()
ethr_atomic_add_acqb()
ethr_atomic_read_bor_acqb()
ethr_atomic_read_band_acqb()
ethr_atomic_xchg_acqb()
ethr_atomic_cmpxchg_acqb()
ethr_atomic_init_relb()
ethr_atomic_set_relb()
ethr_atomic_read_relb()
ethr_atomic_inc_read_relb()
ethr_atomic_dec_read_relb()
ethr_atomic_inc_relb()
ethr_atomic_dec_relb()
ethr_atomic_add_read_relb()
ethr_atomic_add_relb()
ethr_atomic_read_bor_relb()
ethr_atomic_read_band_relb()
ethr_atomic_xchg_relb()
ethr_atomic_cmpxchg_relb()
ethr_atomic_init_ddrb()
ethr_atomic_set_ddrb()
ethr_atomic_read_ddrb()
ethr_atomic_inc_read_ddrb()
ethr_atomic_dec_read_ddrb()
ethr_atomic_inc_ddrb()
ethr_atomic_dec_ddrb()
ethr_atomic_add_read_ddrb()
ethr_atomic_add_ddrb()
ethr_atomic_read_bor_ddrb()
ethr_atomic_read_band_ddrb()
ethr_atomic_xchg_ddrb()
ethr_atomic_cmpxchg_ddrb()
ethr_atomic_init_rb()
ethr_atomic_set_rb()
ethr_atomic_read_rb()
ethr_atomic_inc_read_rb()
ethr_atomic_dec_read_rb()
ethr_atomic_inc_rb()
ethr_atomic_dec_rb()
ethr_atomic_add_read_rb()
ethr_atomic_add_rb()
ethr_atomic_read_bor_rb()
ethr_atomic_read_band_rb()
ethr_atomic_xchg_rb()
ethr_atomic_cmpxchg_rb()
ethr_atomic_init_wb()
ethr_atomic_set_wb()
ethr_atomic_read_wb()
ethr_atomic_inc_read_wb()
ethr_atomic_dec_read_wb()
ethr_atomic_inc_wb()
ethr_atomic_dec_wb()
ethr_atomic_add_read_wb()
ethr_atomic_add_wb()
ethr_atomic_read_bor_wb()
ethr_atomic_read_band_wb()
ethr_atomic_xchg_wb()
ethr_atomic_cmpxchg_wb()

## Double word size

ethr_dw_atomic_init()
ethr_dw_atomic_set()
ethr_dw_atomic_read()
ethr_dw_atomic_cmpxchg()
ethr_dw_atomic_init_mb()
ethr_dw_atomic_set_mb()
ethr_dw_atomic_read_mb()
ethr_dw_atomic_cmpxchg_mb()
ethr_dw_atomic_init_acqb()
ethr_dw_atomic_set_acqb()
ethr_dw_atomic_read_acqb()
ethr_dw_atomic_cmpxchg_acqb()
ethr_dw_atomic_init_relb()
ethr_dw_atomic_set_relb()
ethr_dw_atomic_read_relb()
ethr_dw_atomic_cmpxchg_relb()
ethr_dw_atomic_init_ddrb()
ethr_dw_atomic_set_ddrb()
ethr_dw_atomic_read_ddrb()
ethr_dw_atomic_cmpxchg_ddrb()
ethr_dw_atomic_init_rb()
ethr_dw_atomic_set_rb()
ethr_dw_atomic_read_rb()
ethr_dw_atomic_cmpxchg_rb
ethr_dw_atomic_init_wb()
ethr_dw_atomic_set_wb()
ethr_dw_atomic_read_wb()
ethr_dw_atomic_cmpxchg_wb()



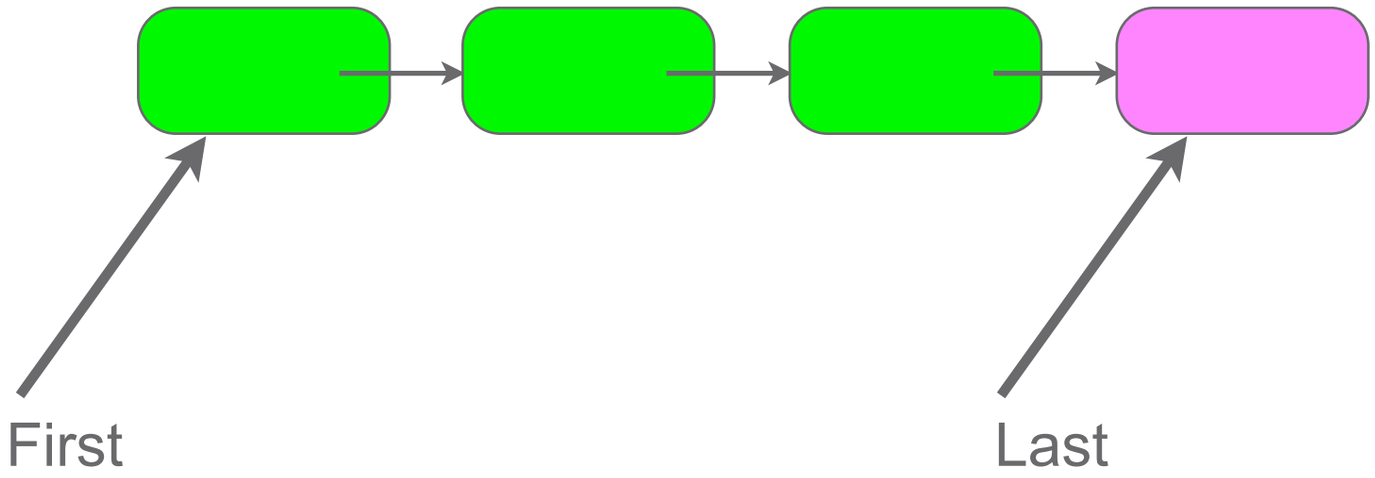
# LOCKED QUEUE



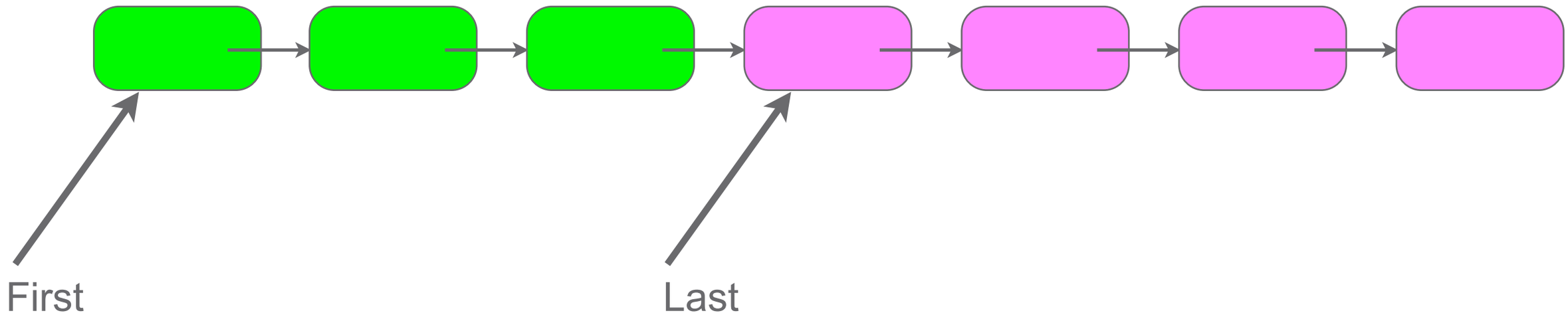
First

Last

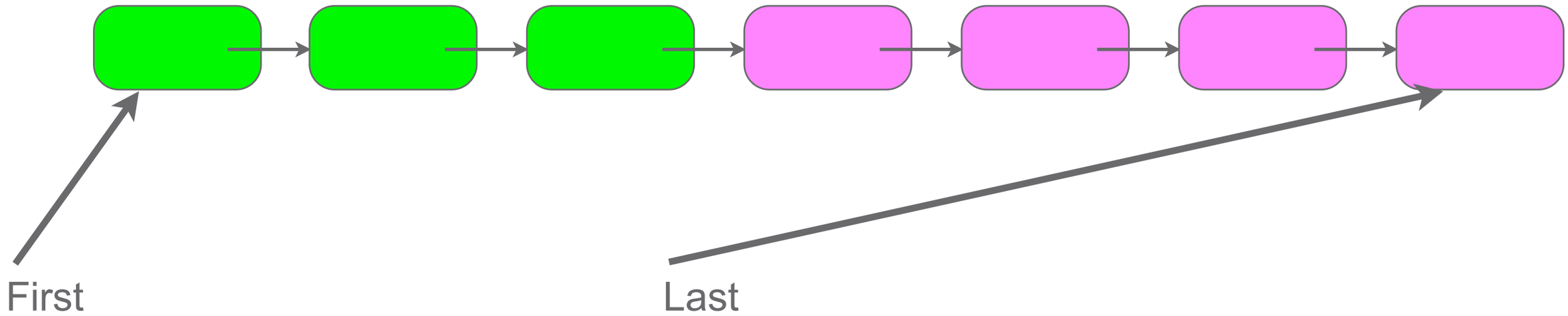
# LOCK FREE QUEUE



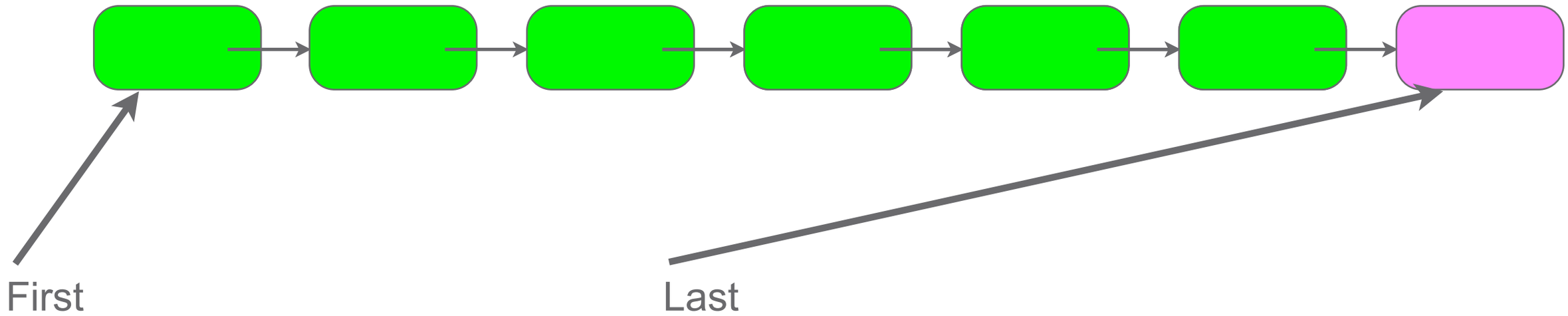
# LOCK FREE QUEUE



# LOCK FREE QUEUE

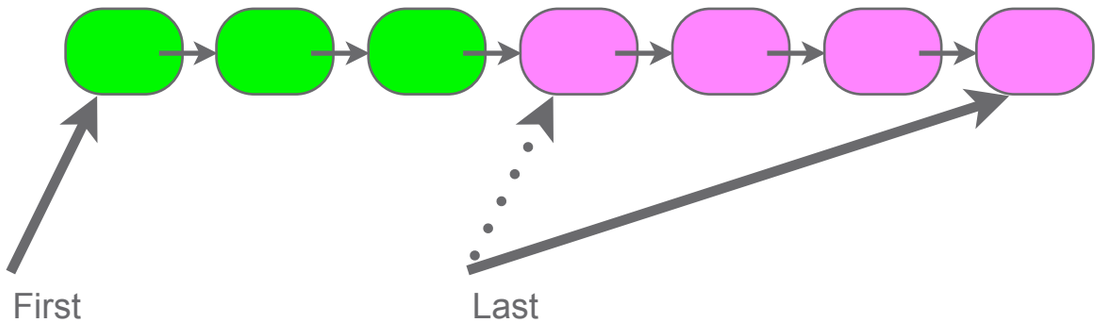


# LOCK FREE QUEUE



# THREAD PROGRESS

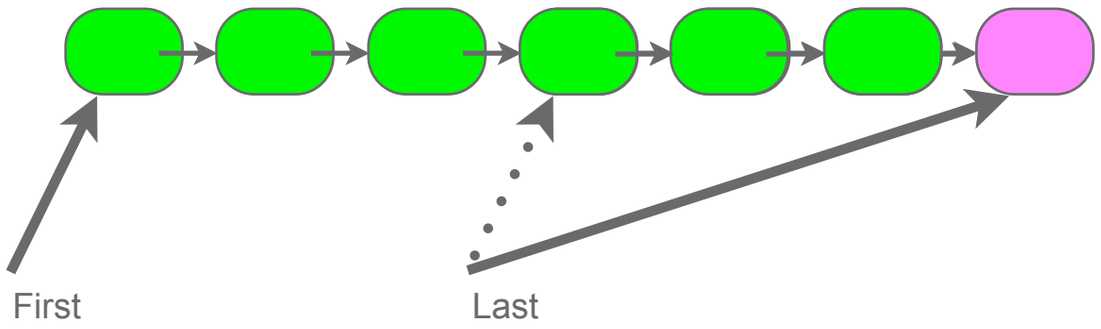
- We want a mechanism that can be used in order to determine when all “interesting” threads
  - have at least once returned to the event loop code
  - have at least once executed a full memory barrier
  - will see all modifications I have made
- Interesting threads are “managed”:
  - We know about all of them
  - They do not hang
  - They will always return to a certain point
  - Typically - the schedulers and a few other threads




- The mechanism is called Thread Progress in the VM code
- Unmanaged threads need to be taken care of by other means

# THREAD PROGRESS

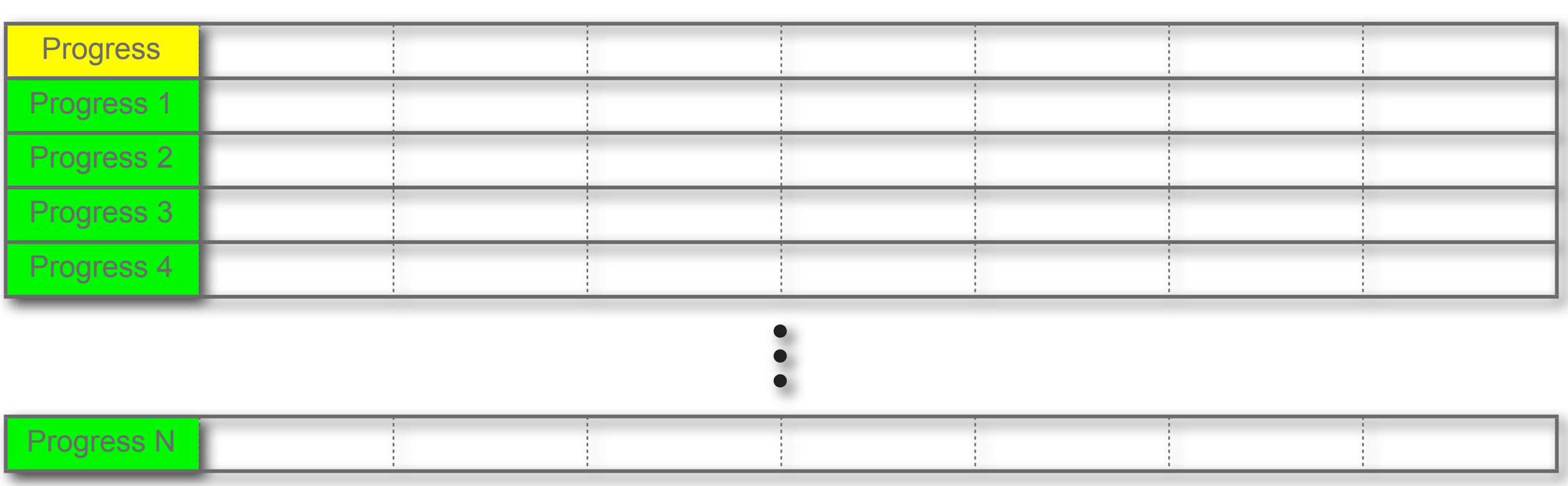
- We want a mechanism that can be used in order to determine when all “interesting” threads
  - have at least once returned to the event loop code
  - have at least once executed a full memory barrier
  - will see all modifications I have made
- Interesting threads are “managed”:
  - We know about all of them
  - They do not hang
  - They will always return to a certain point
  - Typically - the schedulers and a few other threads




- The mechanism is called Thread Progress in the VM code
- Unmanaged threads need to be taken care of by other means

# THREAD PROGRESS

- Master thread coordinates
- Thread progress communicated via thread specific cache lines

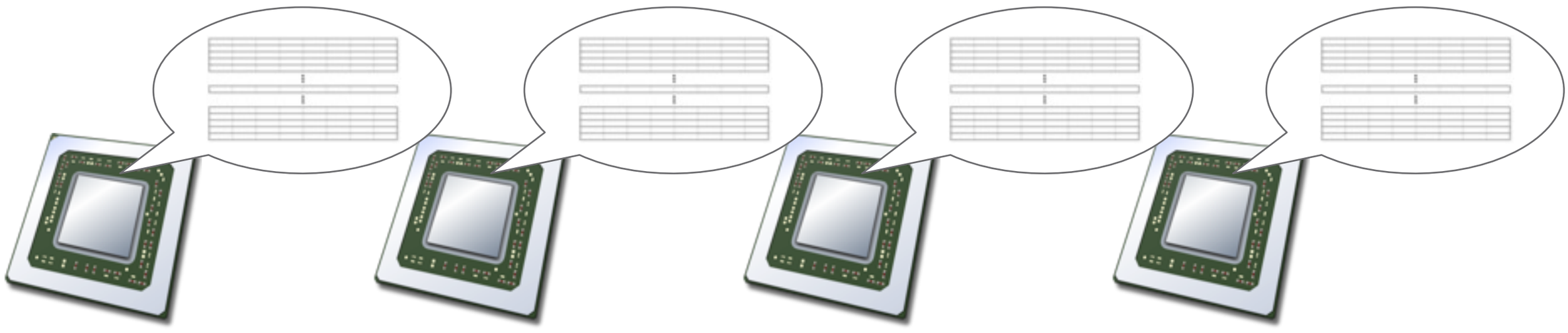




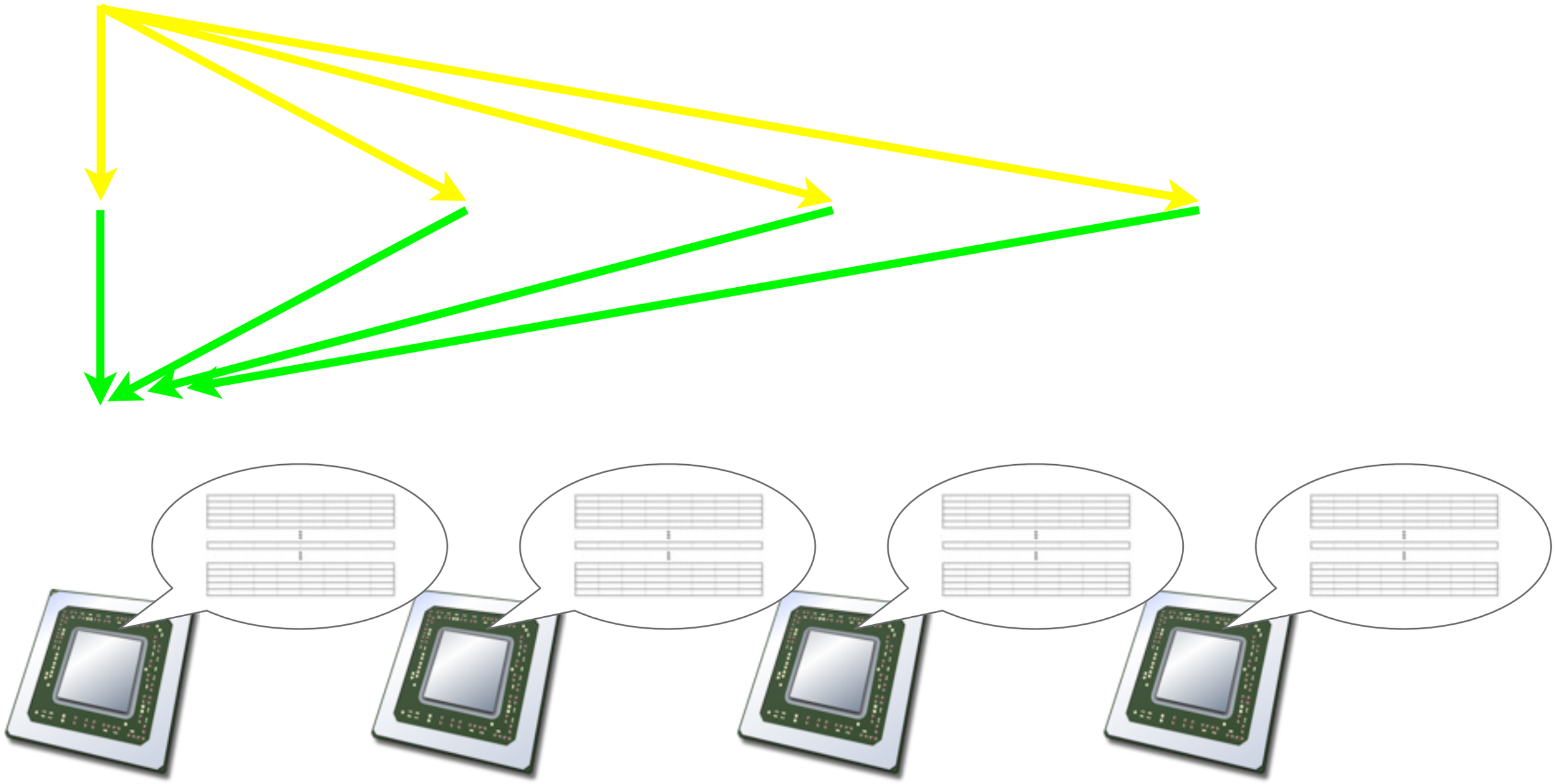

ERLANG

# THREAD PROGRESS

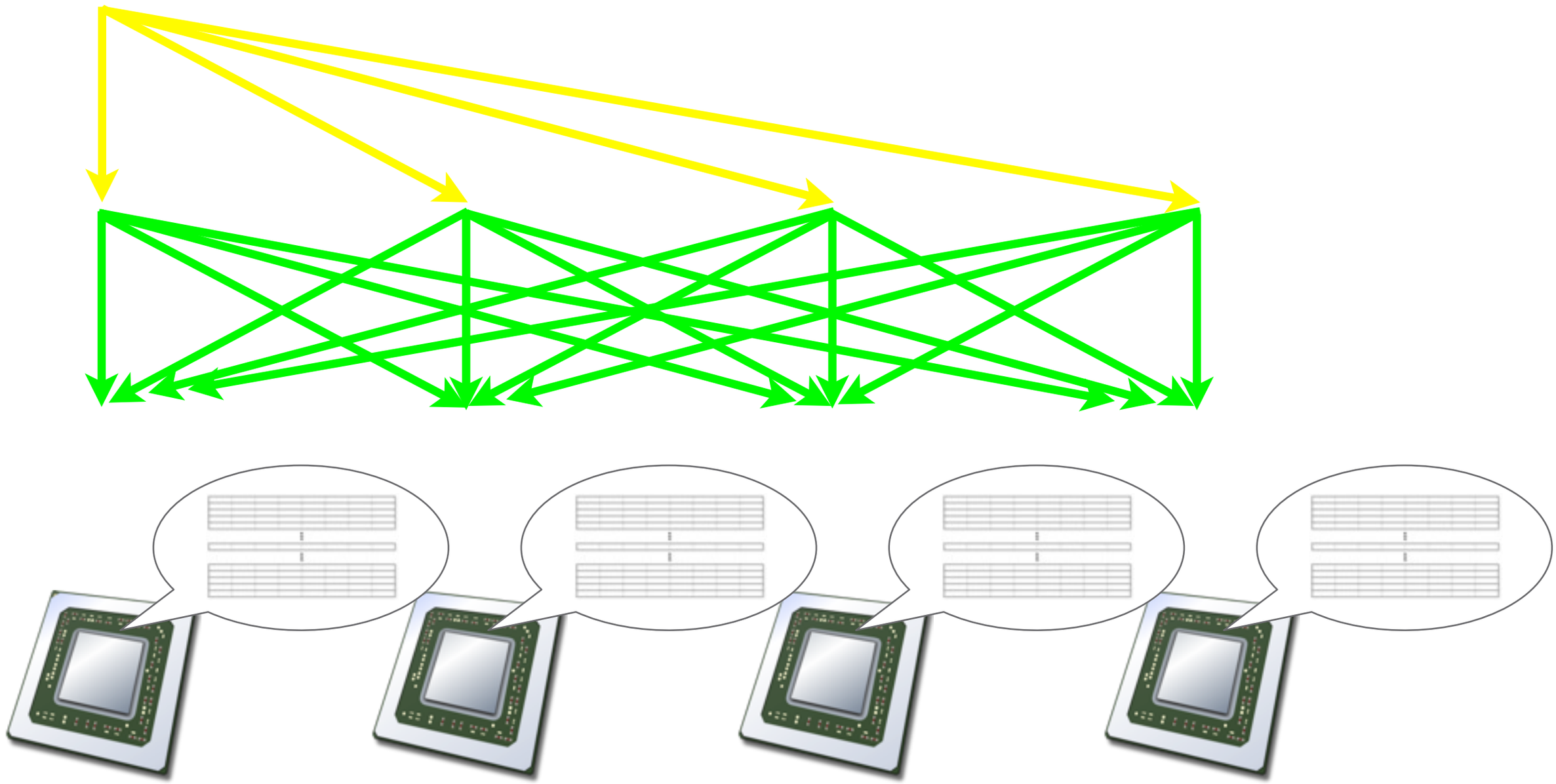




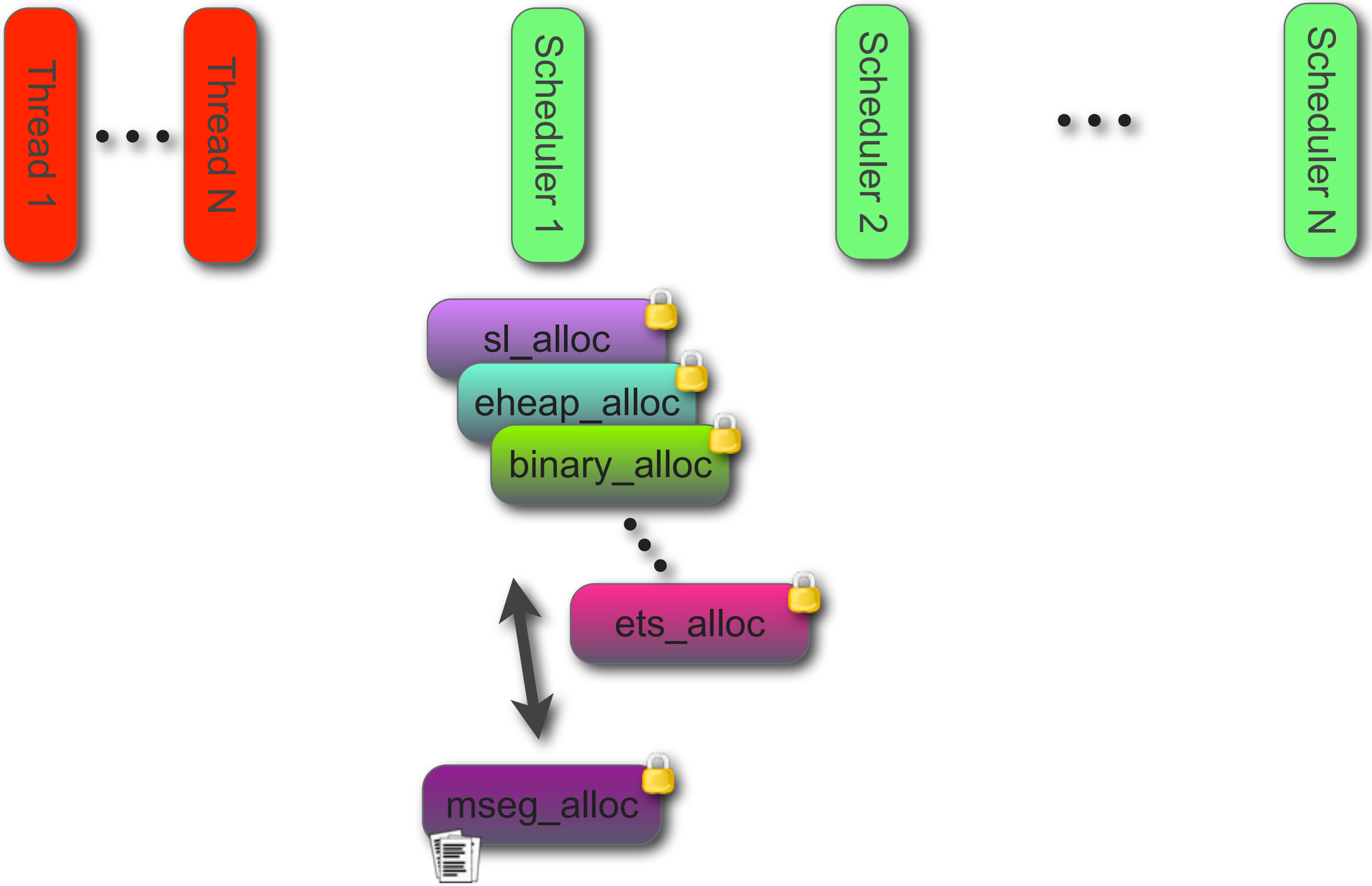
# THREAD PROGRESS



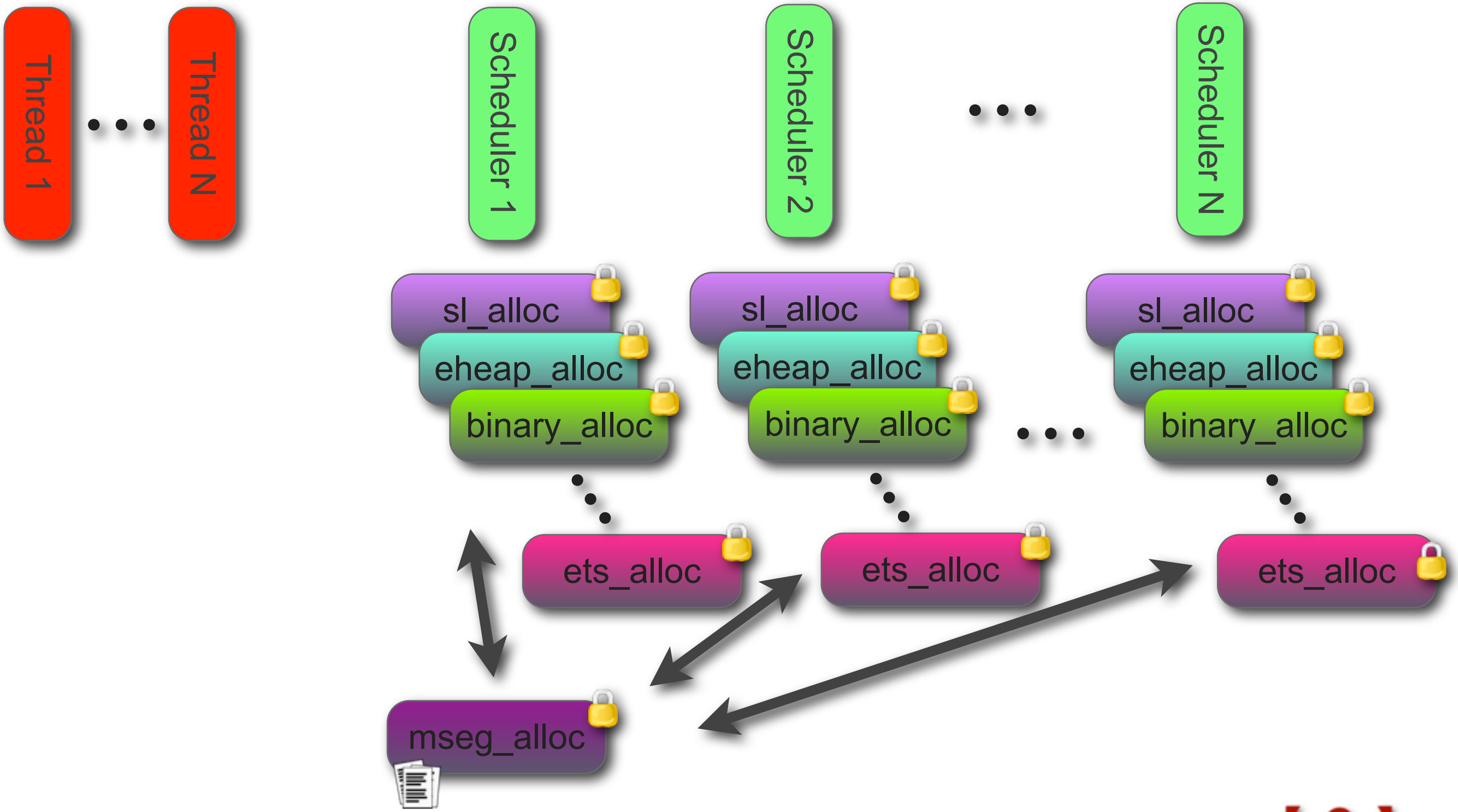
# THREAD PROGRESS



# MEMORY ALLOCATORS R11B



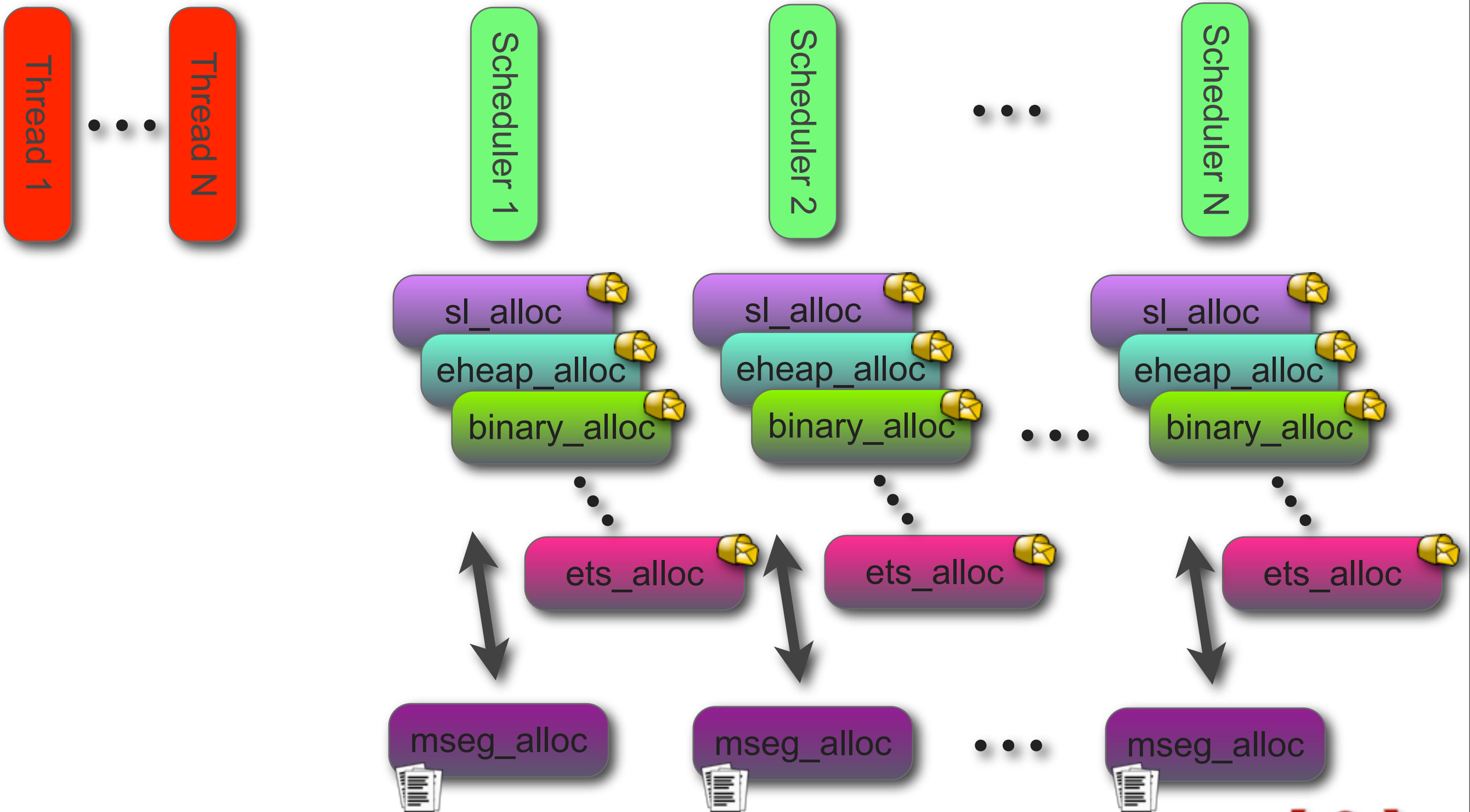
# MEMORY ALLOCATORS R12B-1






ERLANG

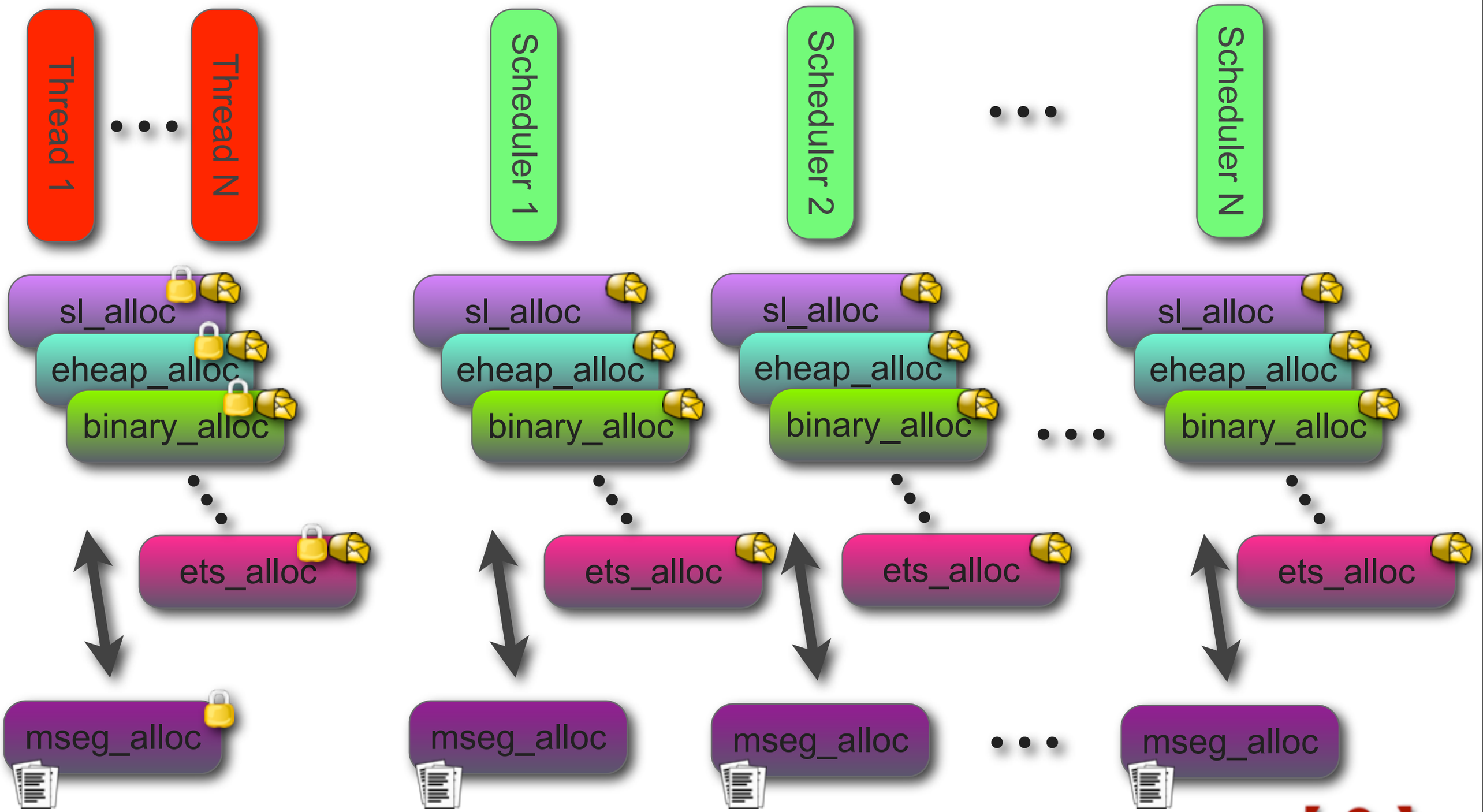
# MEMORY ALLOCATORS R15B






ERLANG

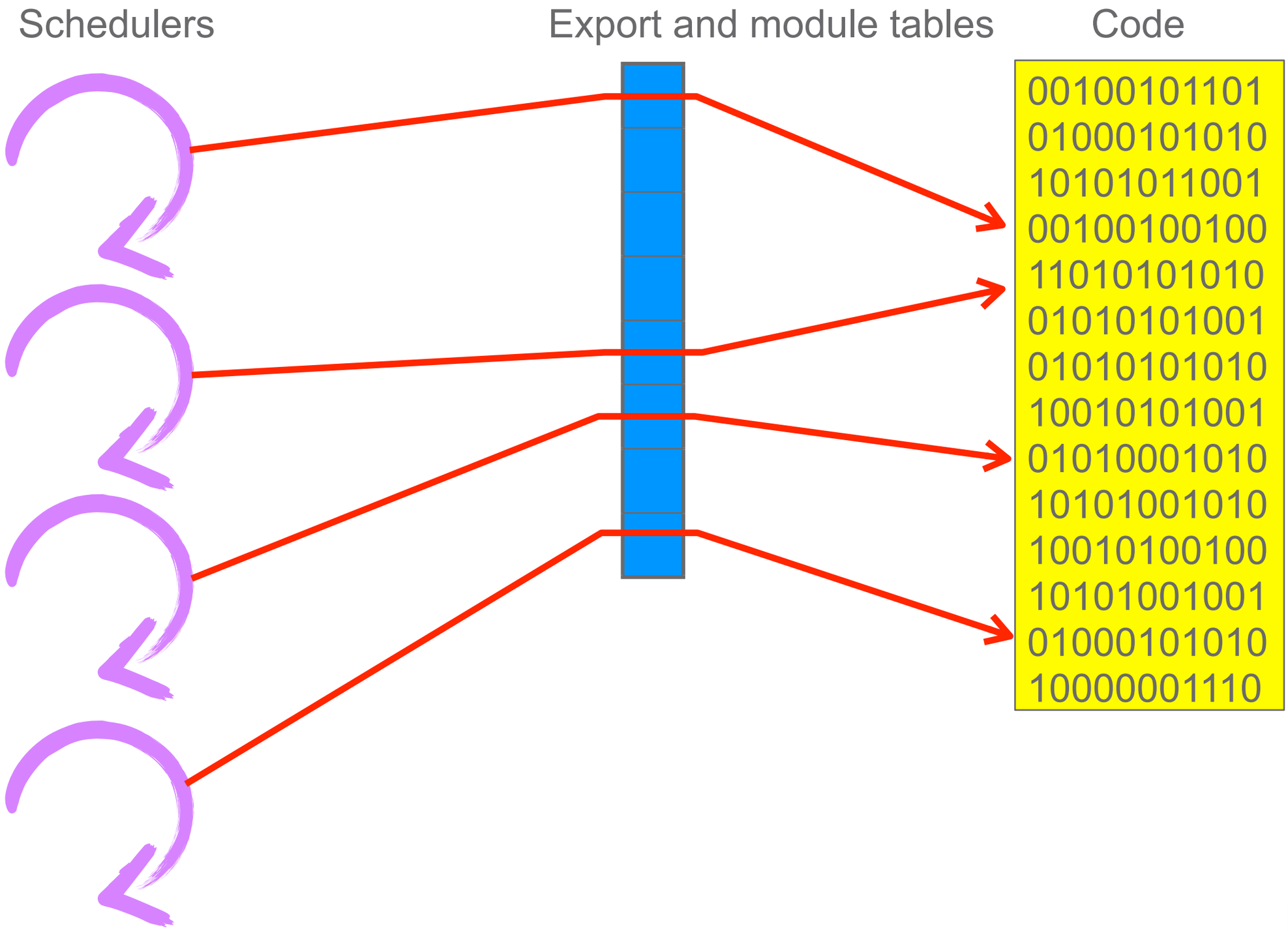
# MEMORY ALLOCATORS R15B






ERLANG

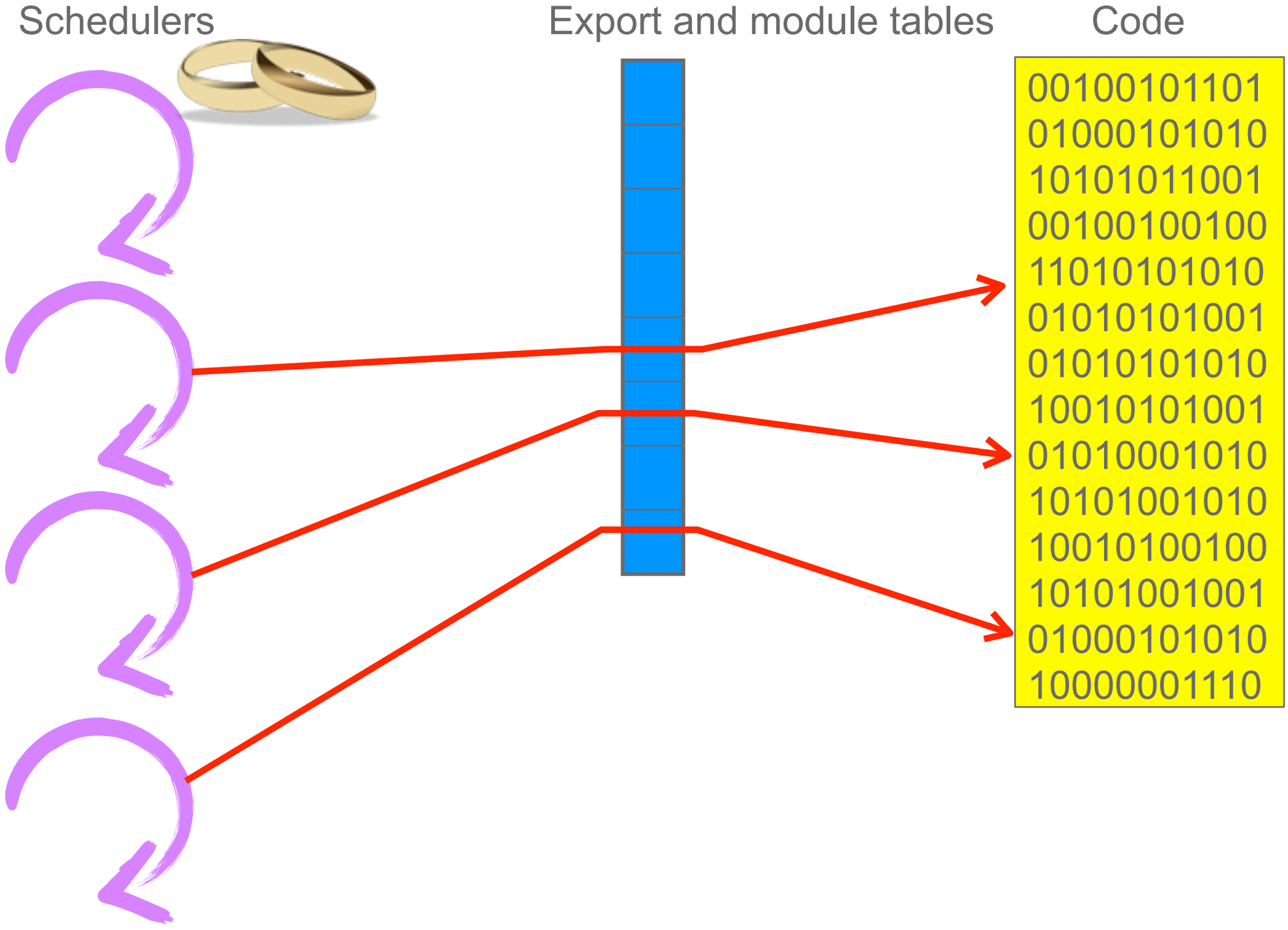
# CODE-LOADING R11-R15



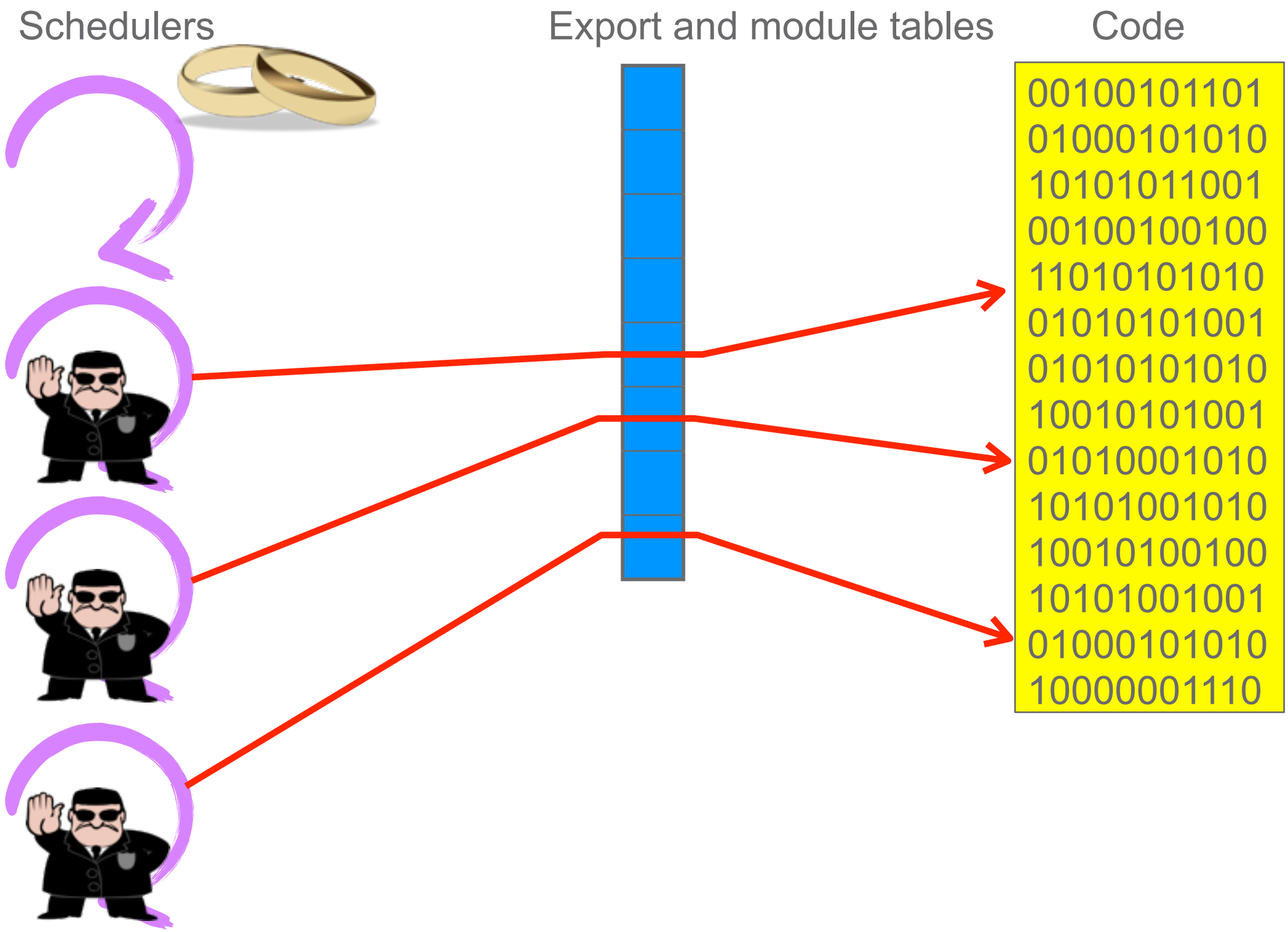



ERLANG

# CODE-LOADING R11-R15

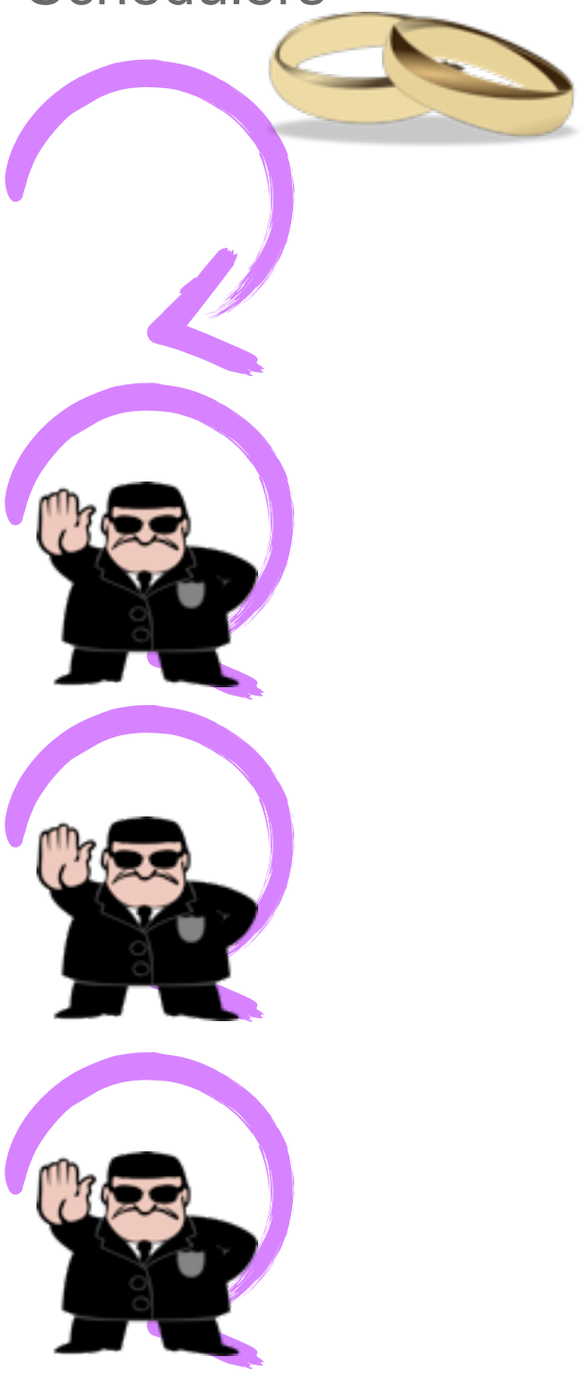


# CODE-LOADING R11-R15

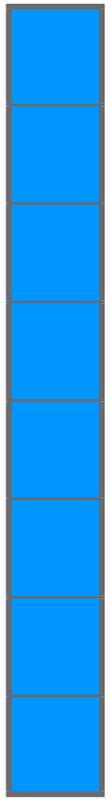


# CODE-LOADING R11-R15

Schedulers



Export and module tables

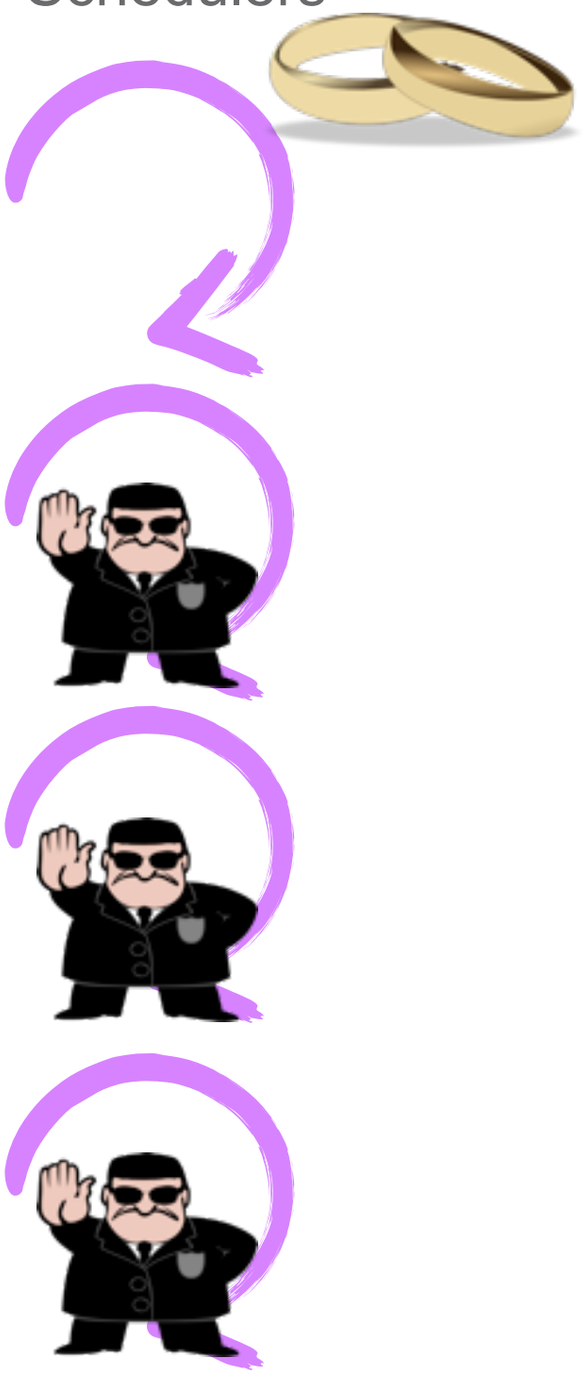


Code

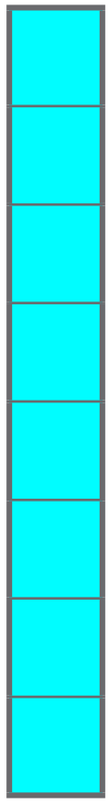
```
00100101101
01000101010
10101011001
00100100100
11010101010
01010101001
01010101010
10010101001
01010001010
10101001010
10010100100
10101001001
01000101010
10000001110
```

# CODE-LOADING R11-R15

Schedulers



Export and module tables



Code

```
00100101101
01000101010
10101011001
00100100100
11010101010
01010101001
01010101010
10010101001
01010001010
10101001010
10010100100
10101001001
01000101010
10000001110
```

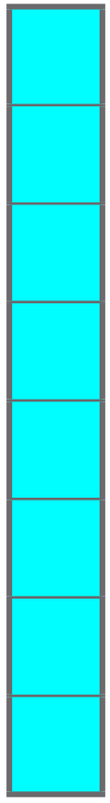


ERLANG

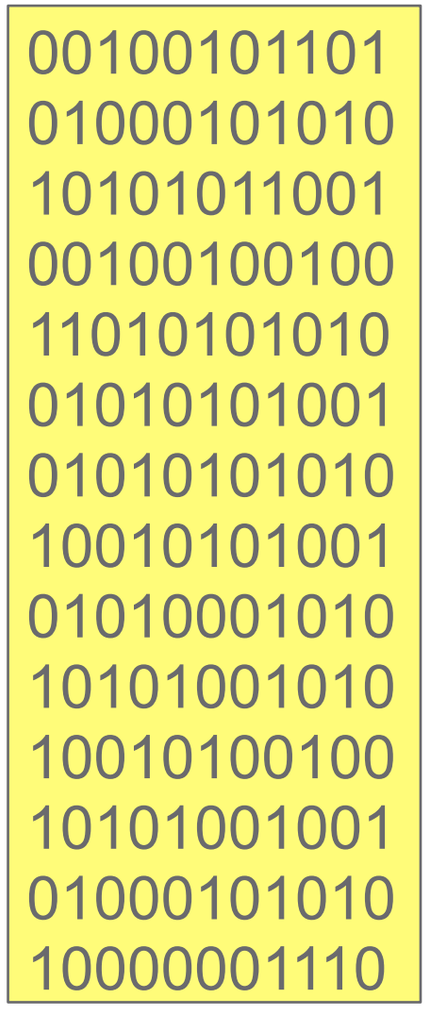
# CODE-LOADING R11-R15

Schedulers

Export and module tables



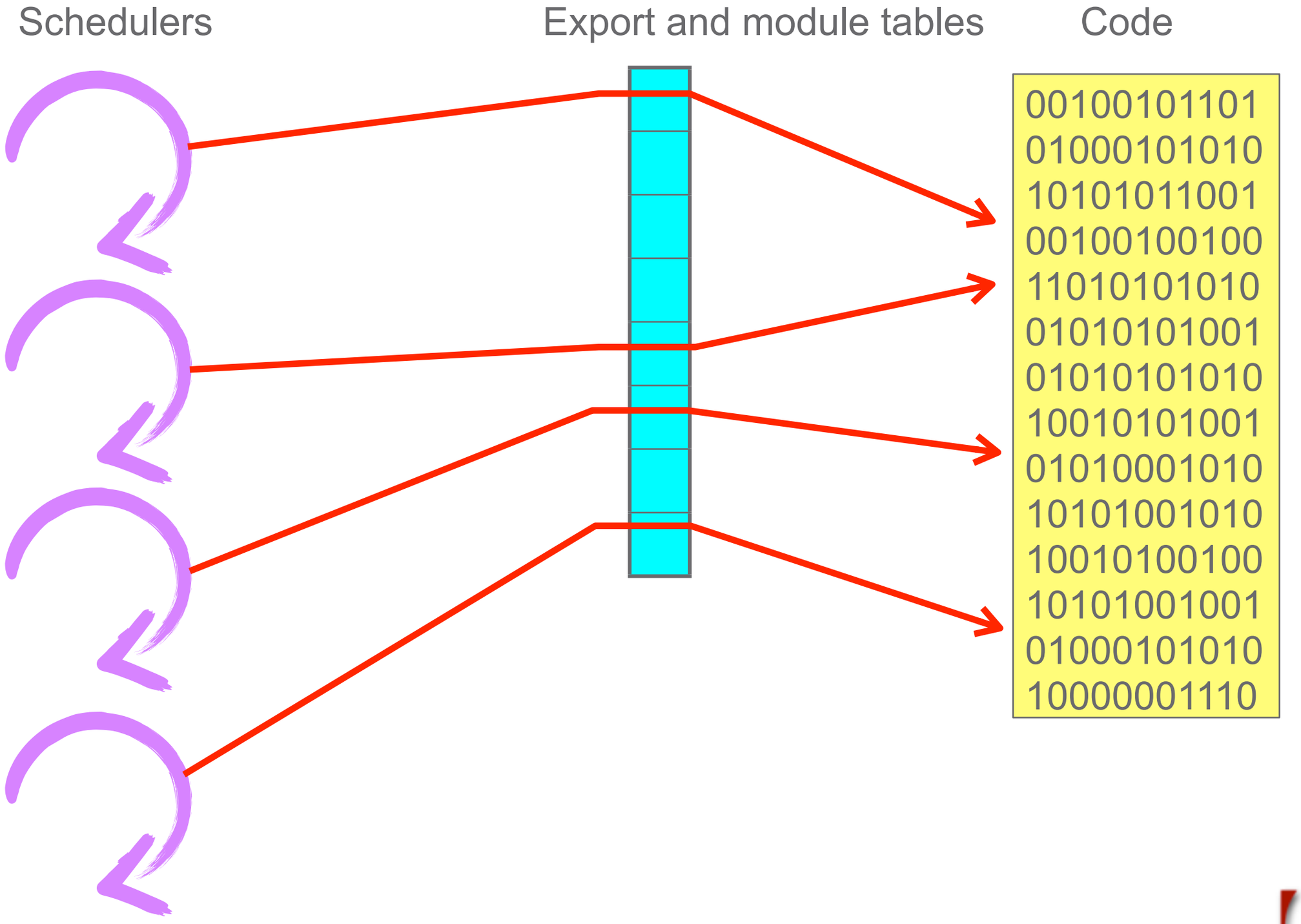
Code



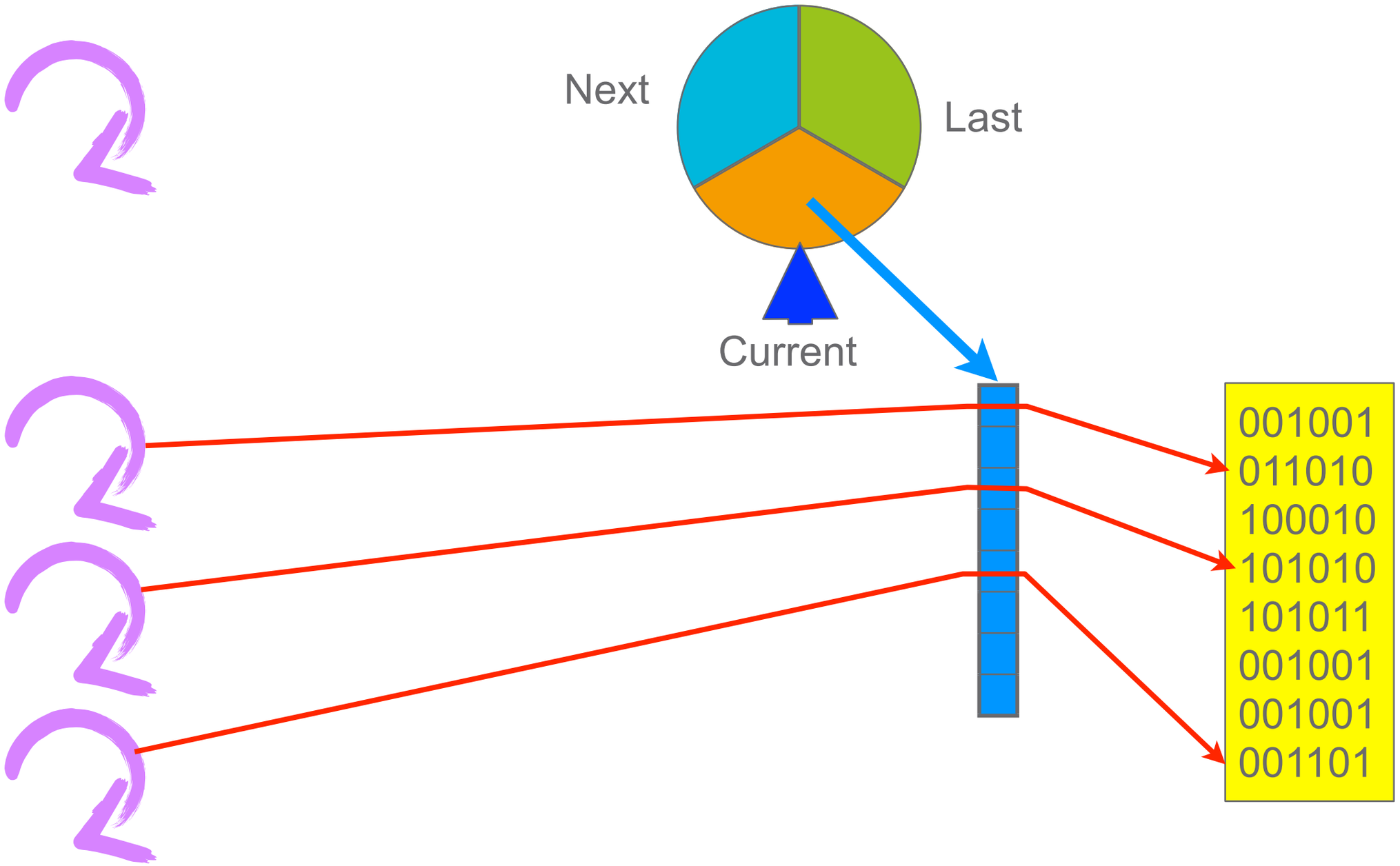



ERLANG

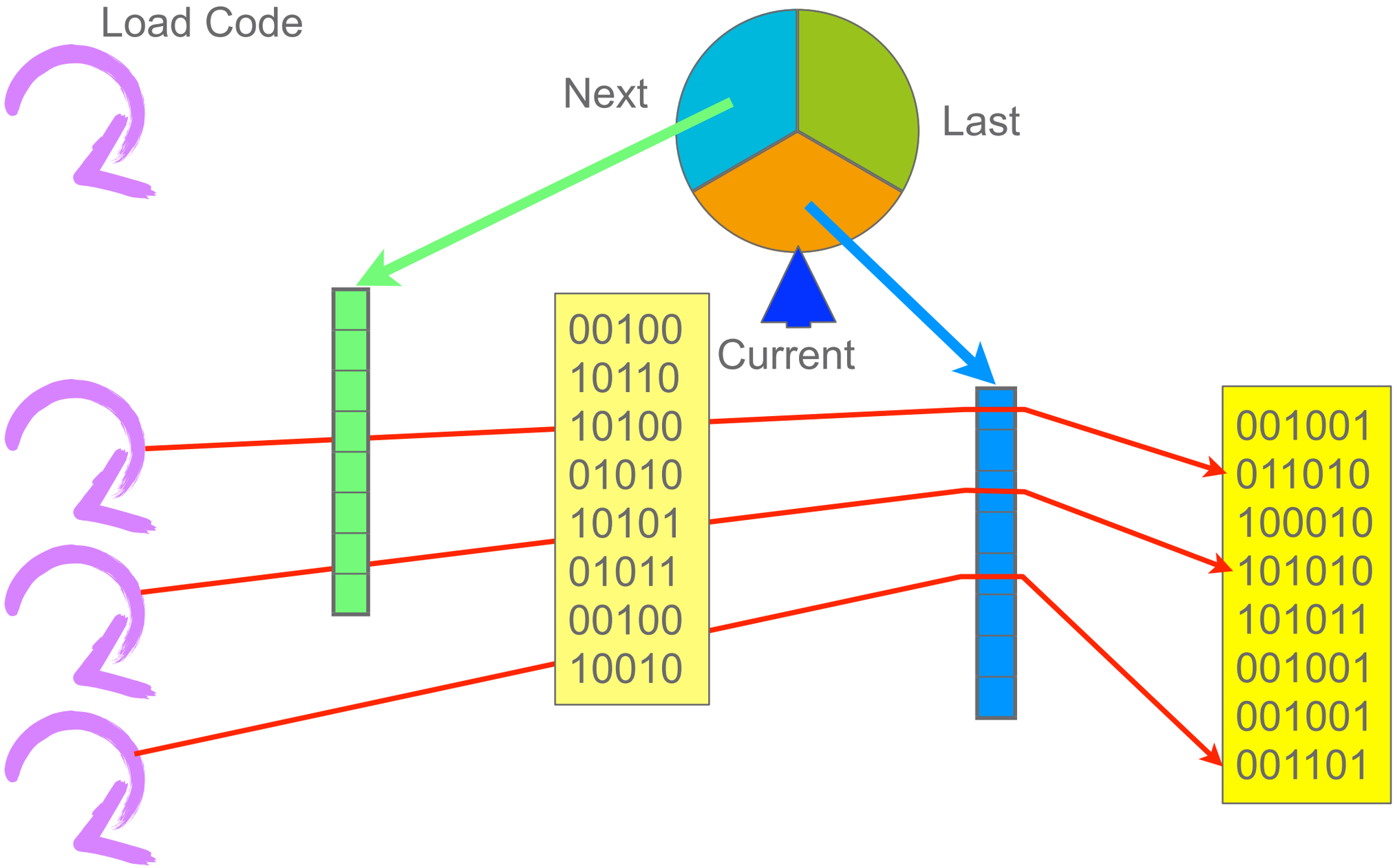
# CODE-LOADING R11-R15



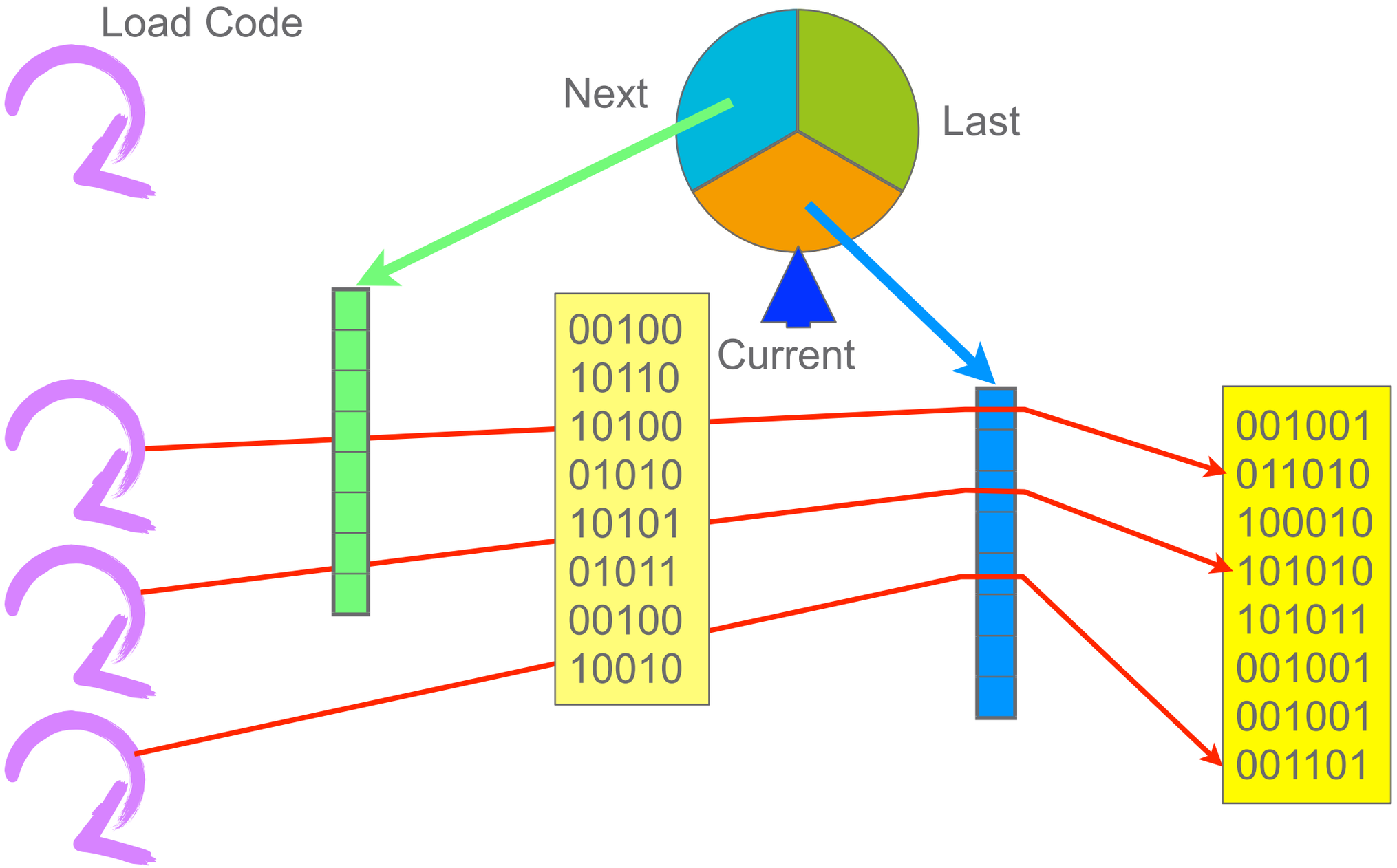
# CODE-LOADING R16



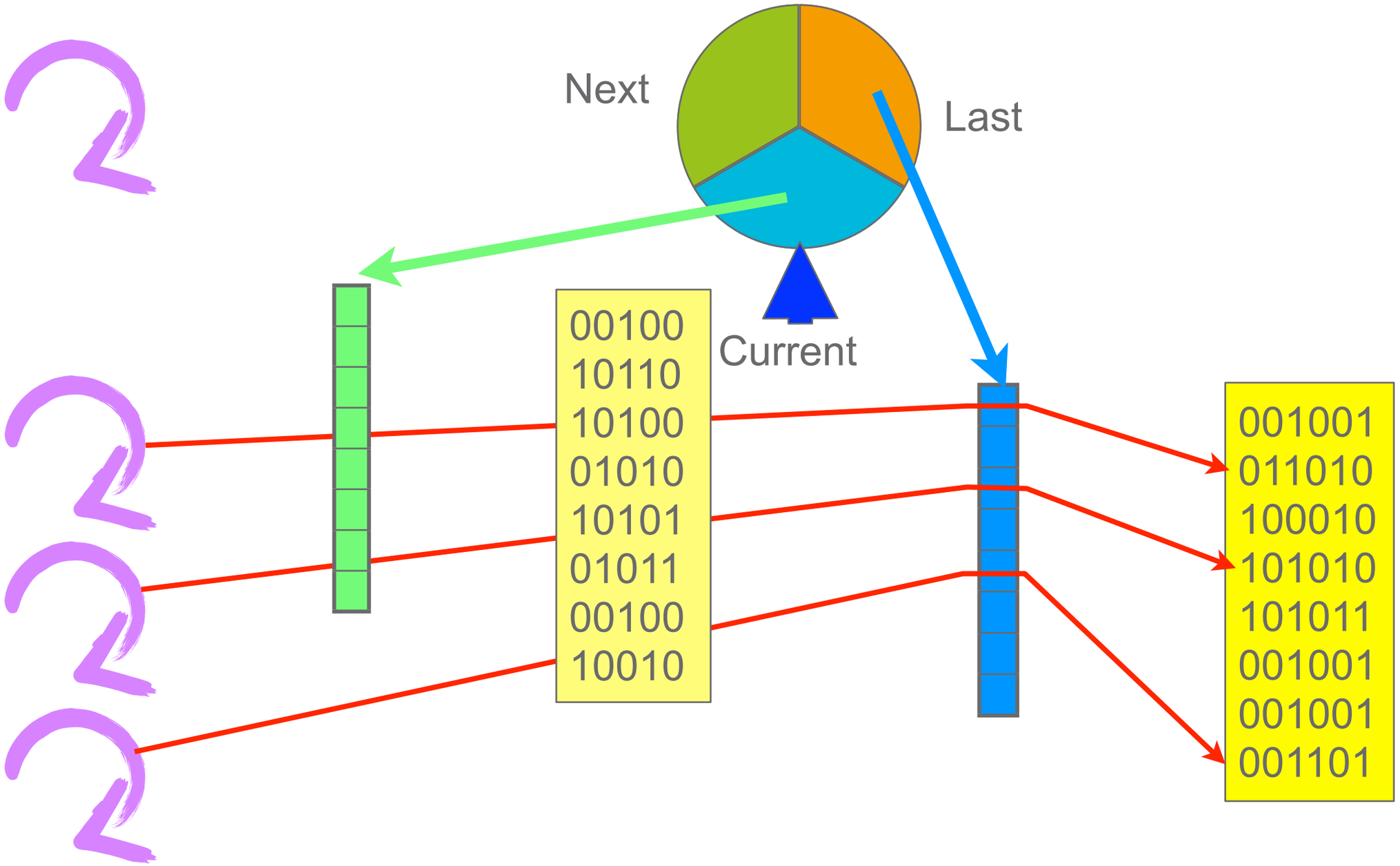
# CODE-LOADING R16



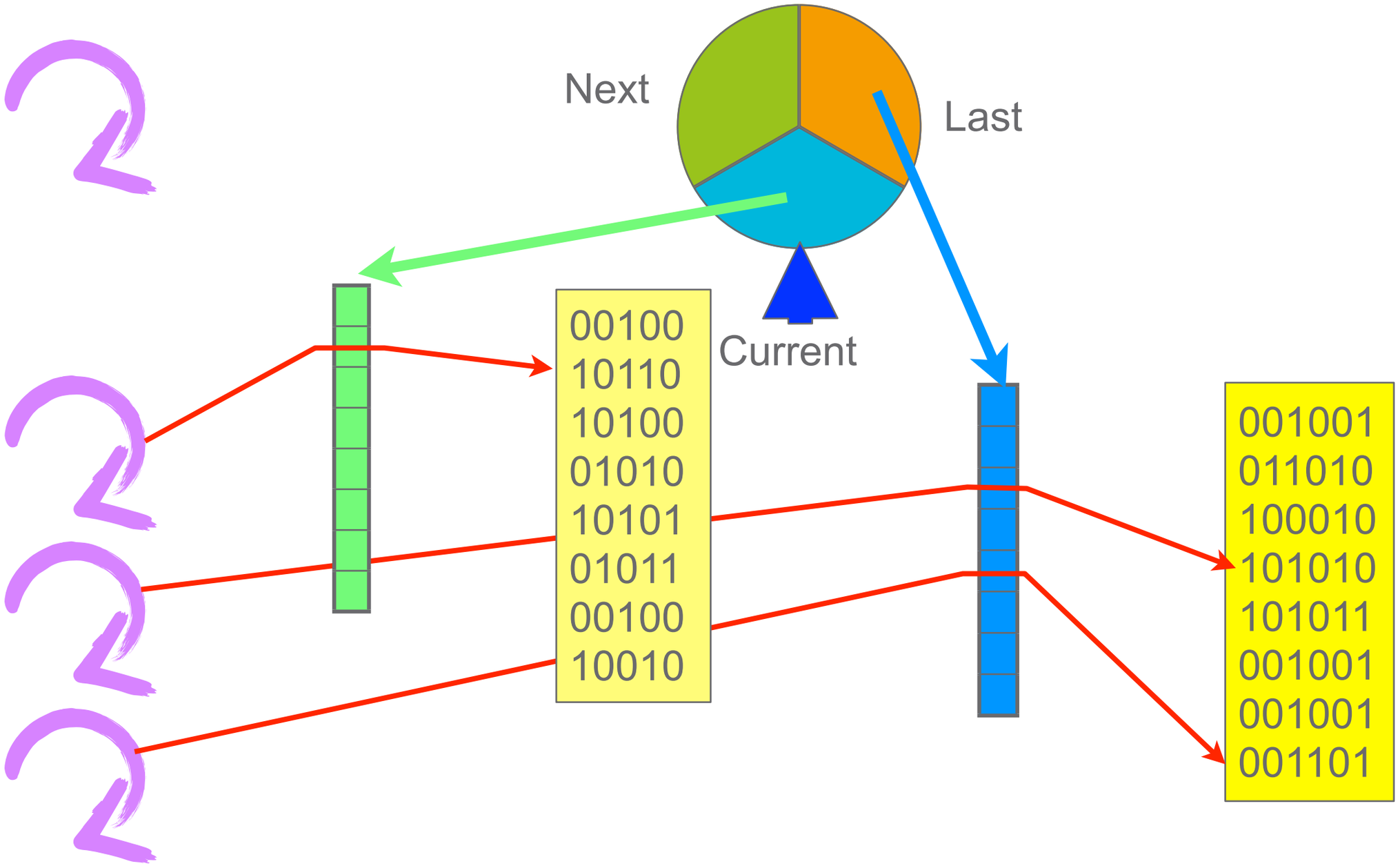
# CODE-LOADING R16



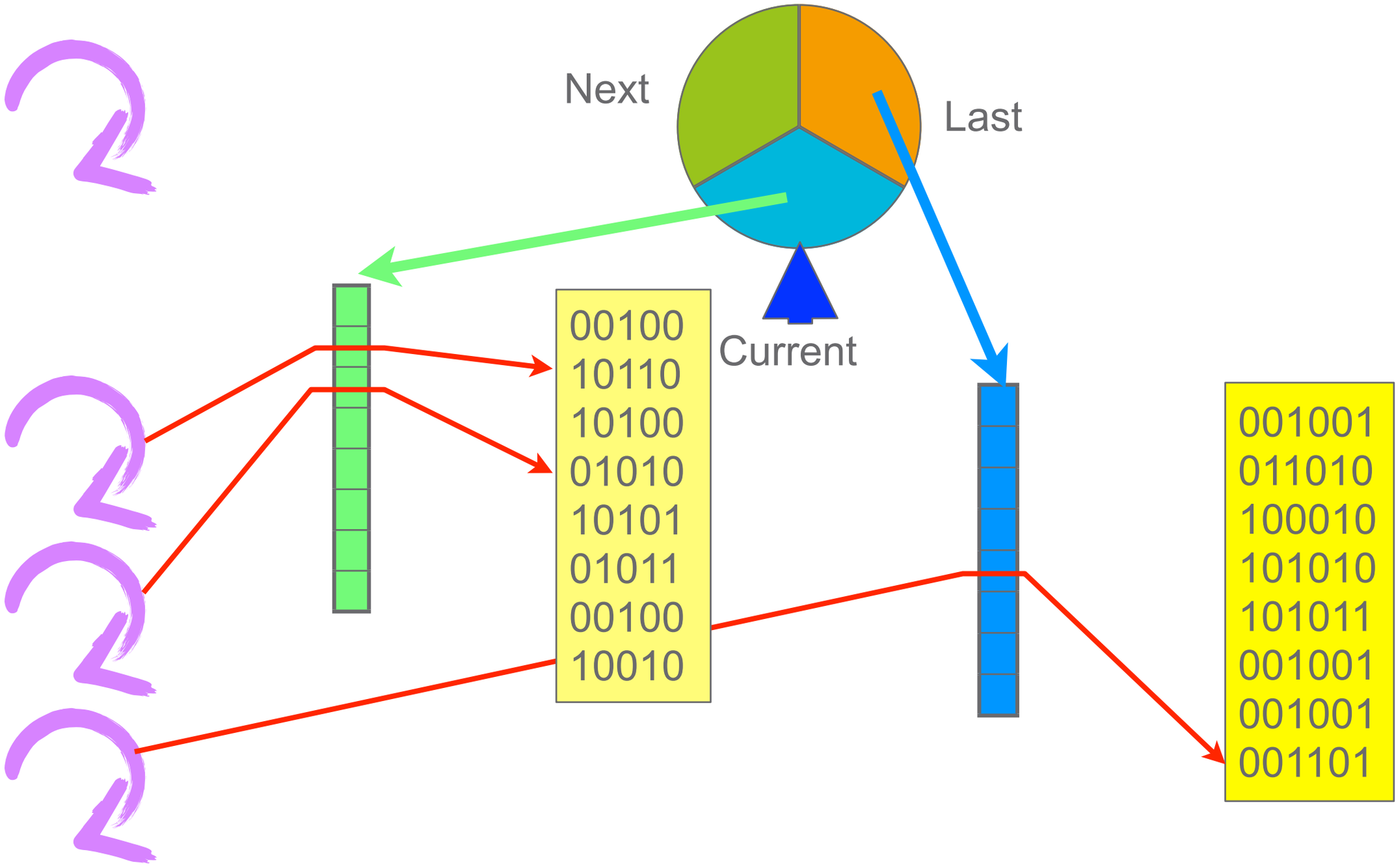
# CODE-LOADING R16



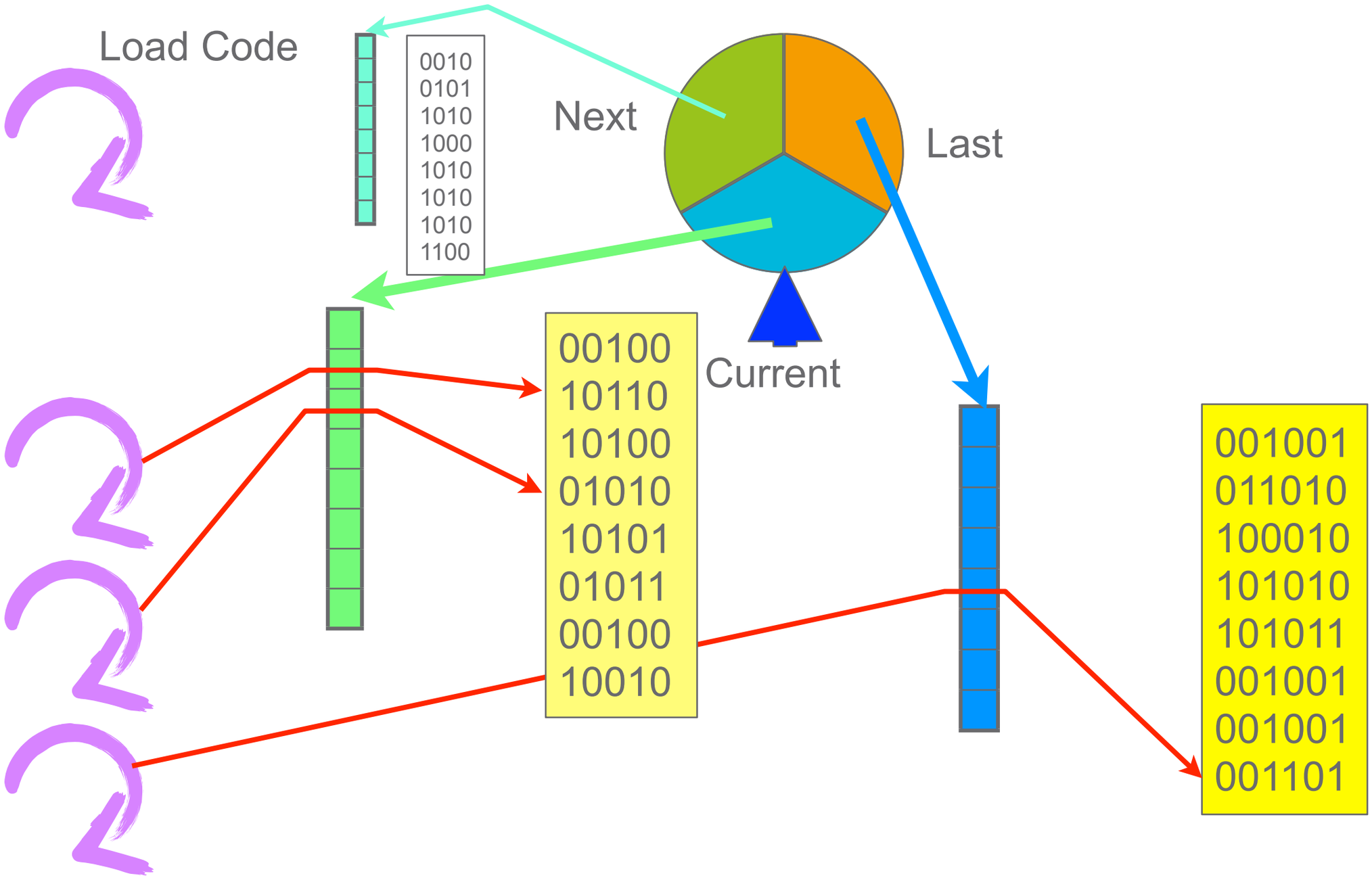
# CODE-LOADING R16



# CODE-LOADING R16



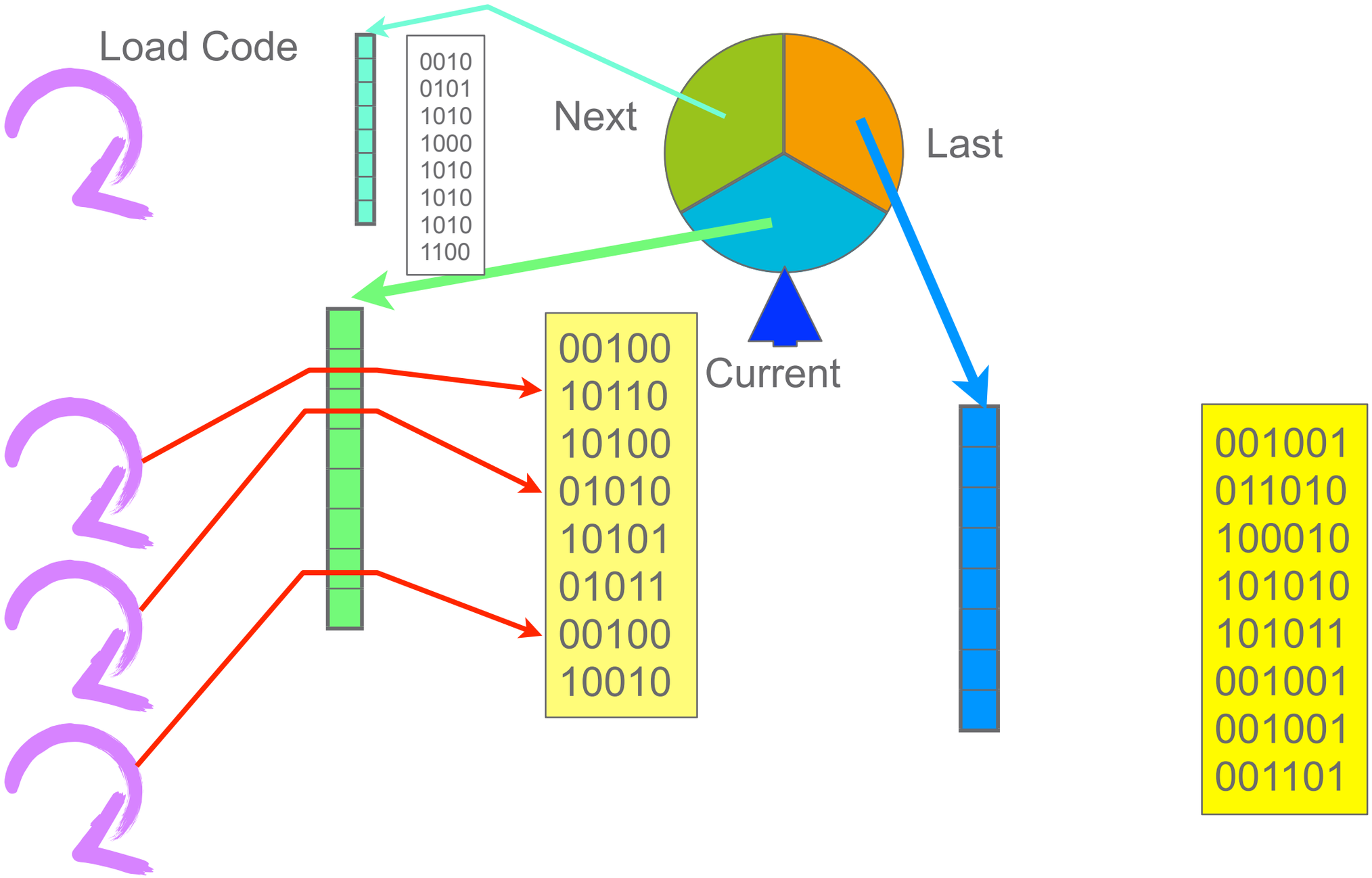
# CODE-LOADING R16






ERLANG

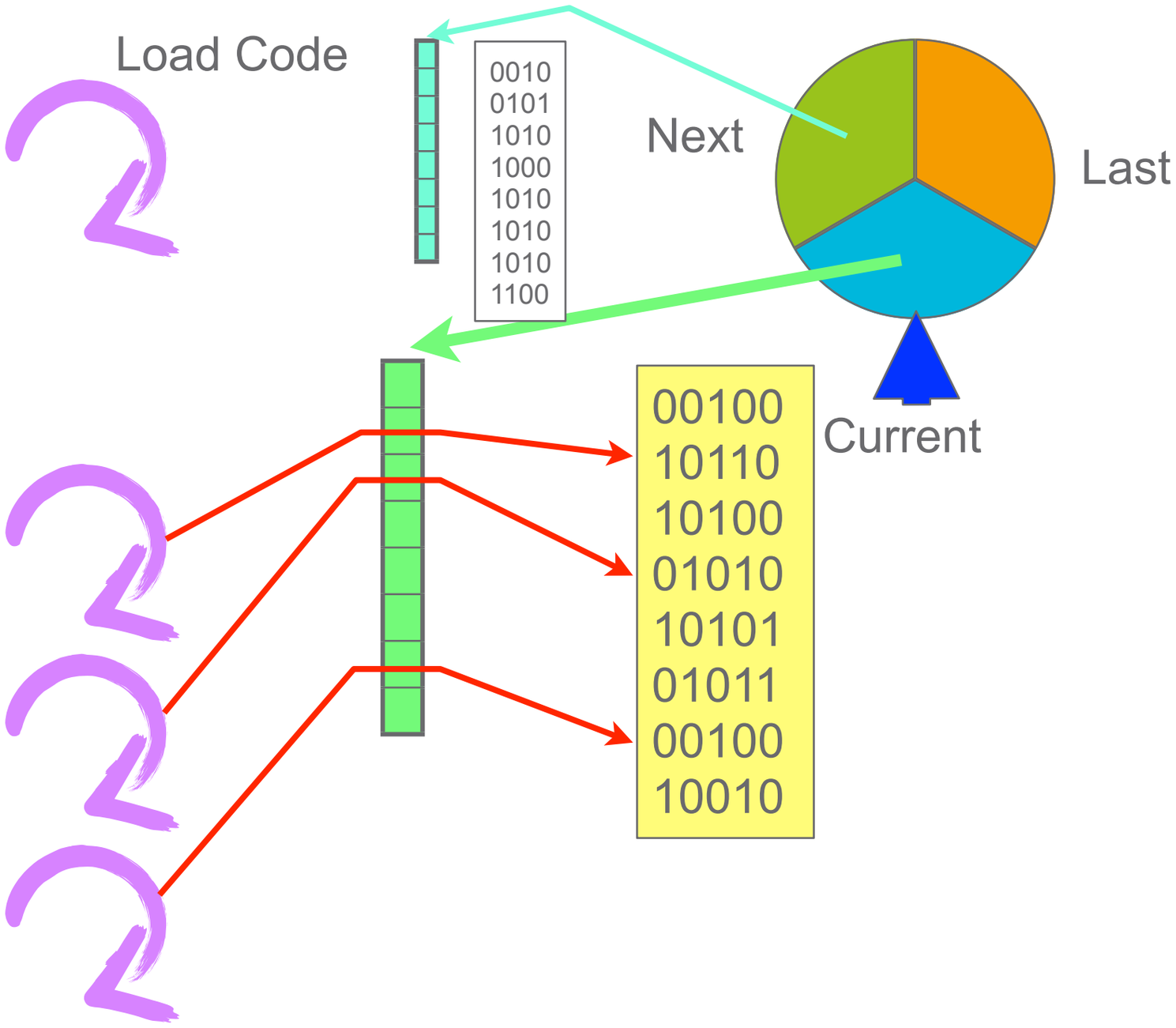
# CODE-LOADING R16



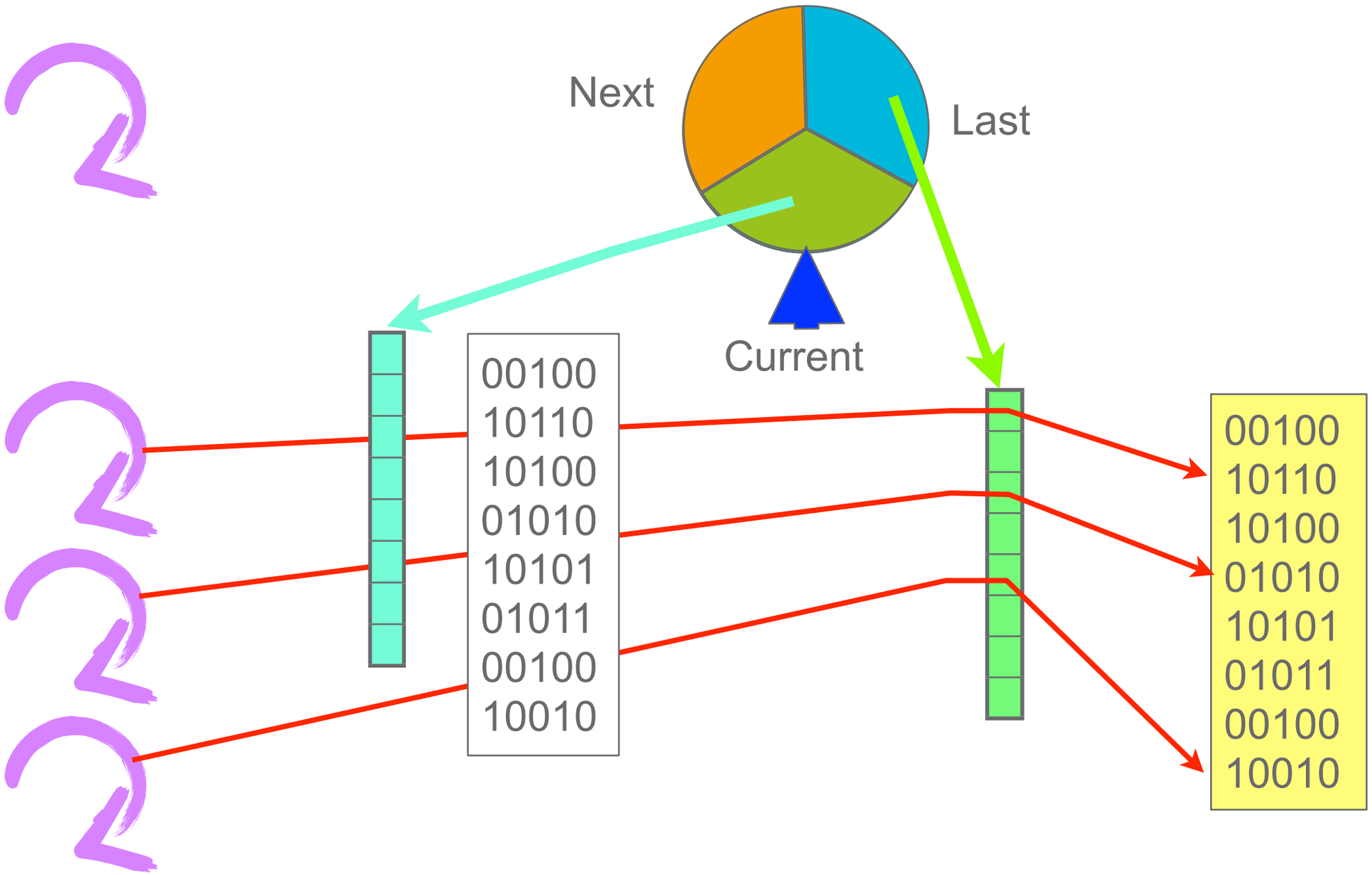



ERLANG

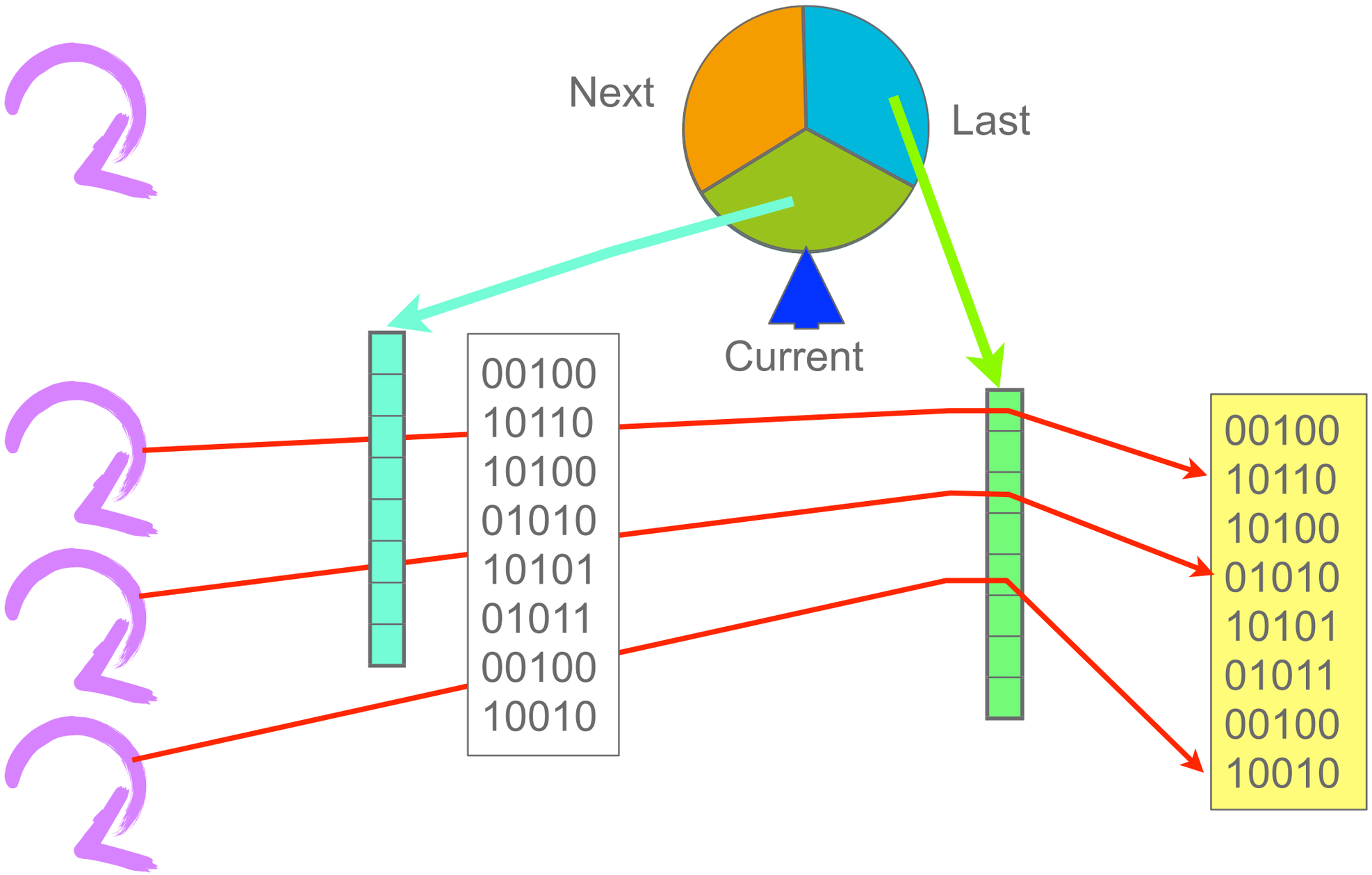
# CODE-LOADING R16



# CODE-LOADING R16



# CODE-LOADING R16



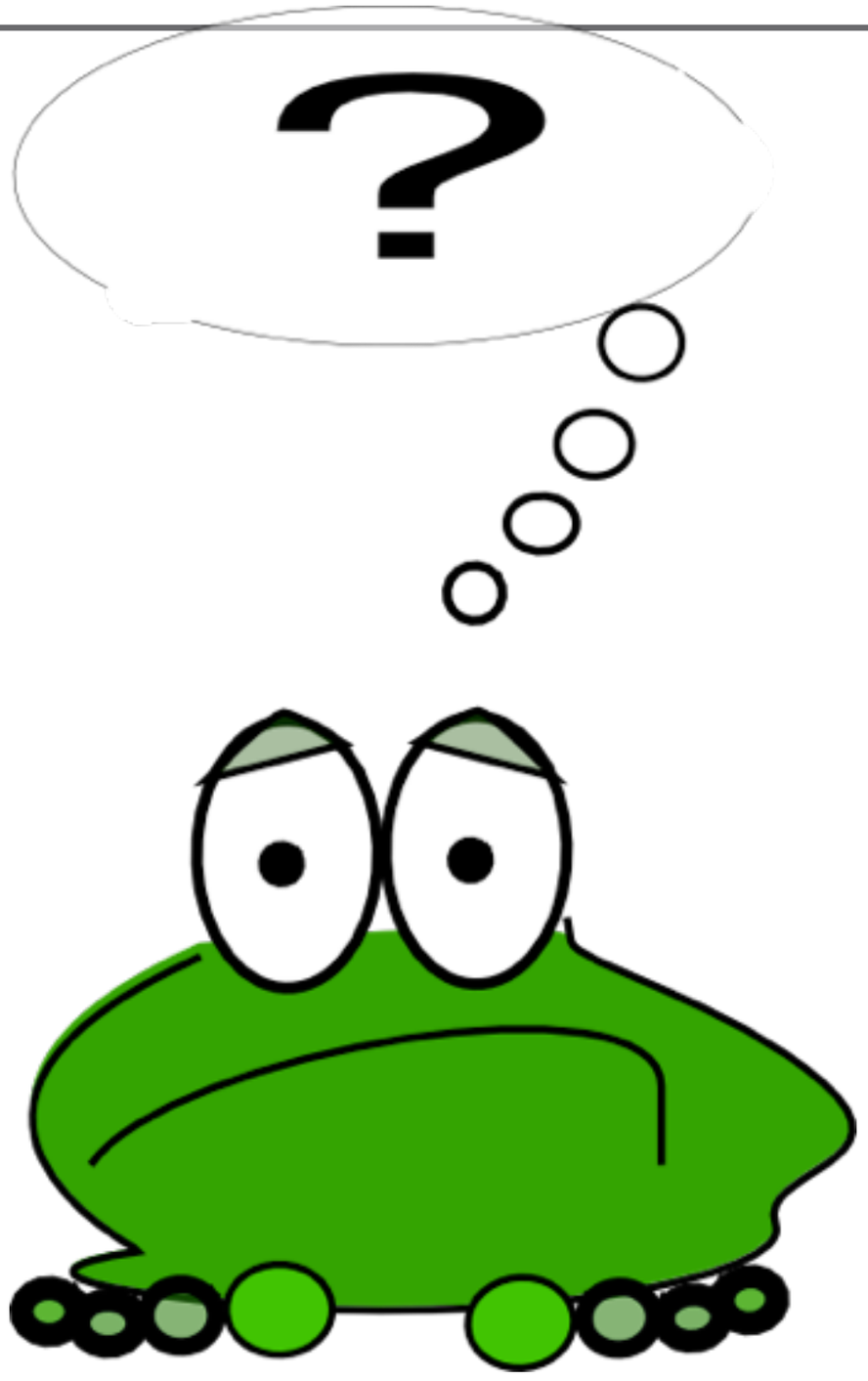
# RECIPES

- Divide or multiply resources
  - Separate memory
  - Separate cache-lines
- Asynchronously schedule work
  - Delayed dealloc
- Only involve relevant parties
  - Fine-grained synchronization
  - Lock-free queues
  - Reader optimized RW-Locks
- Postpone until possible
  - Thread progress
    - Lock-free queues
    - Code loading
  - Table cleanup



ERLANG

# QUESTIONS?



ERICSSON