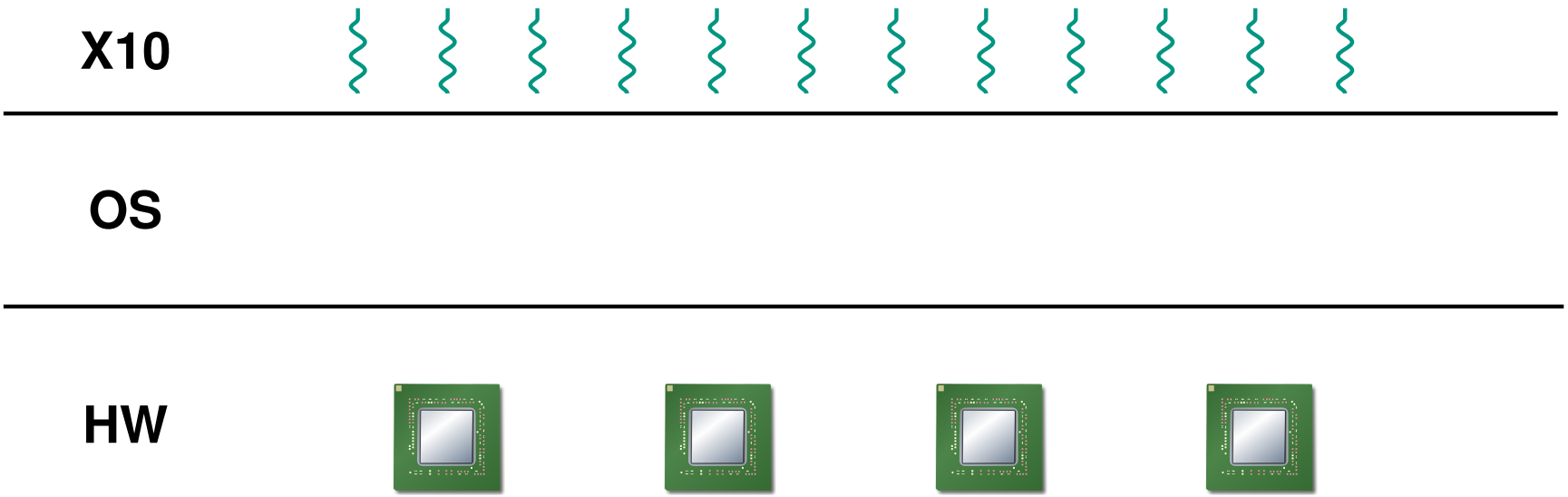
Cutting Out the Middleman: OS-Level Support for X10 Activities

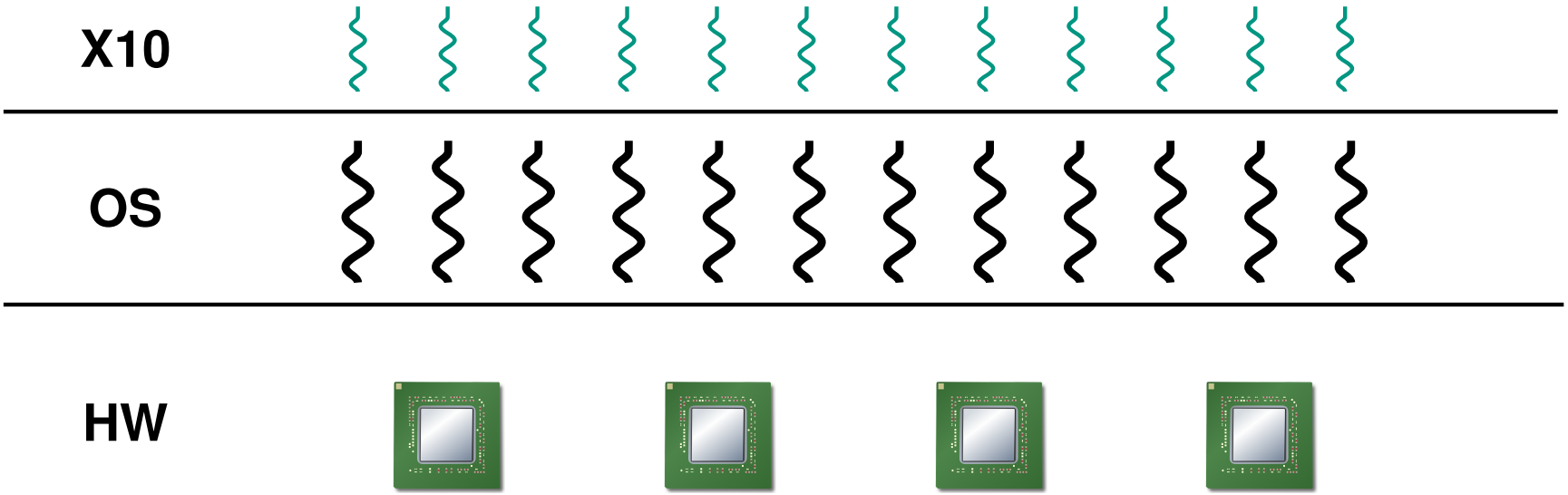
Manuel Mohr, Sebastian Buchwald, Andreas Zwinkau, Christoph Erhardt, Benjamin Oechslein, Jens Schedel, Daniel Lohmann

Chair for Programming Paradigms, Karlsruhe Institute of Technology (KIT) & System Software Group, University Erlangen-Nuremberg (FAU)

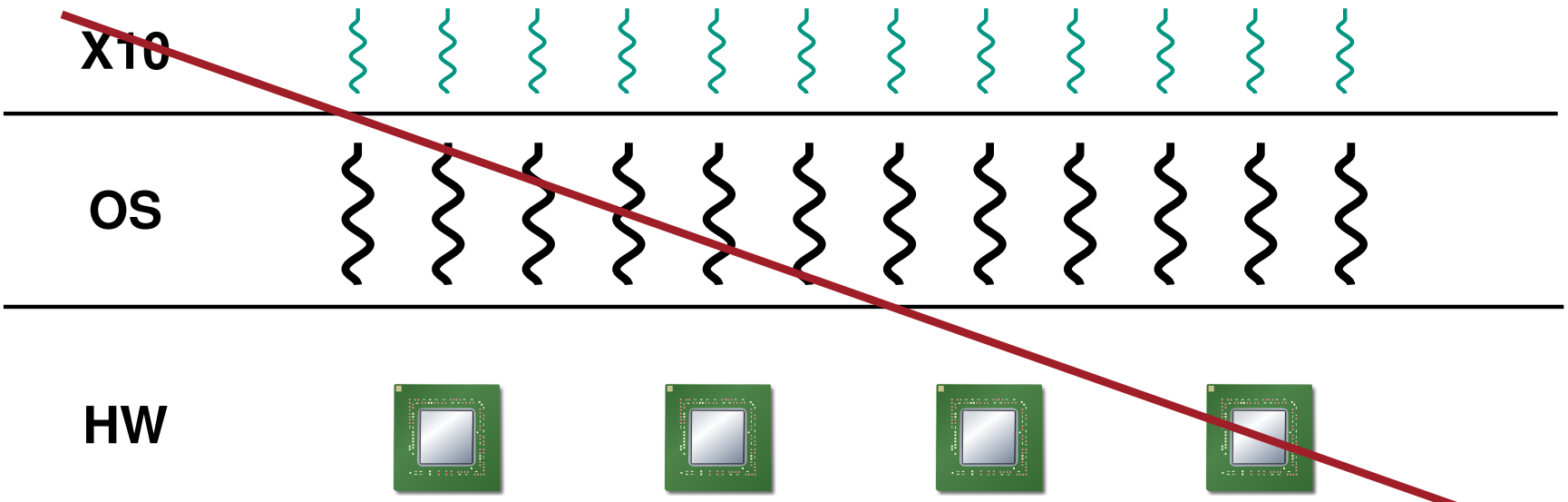
Ideal World



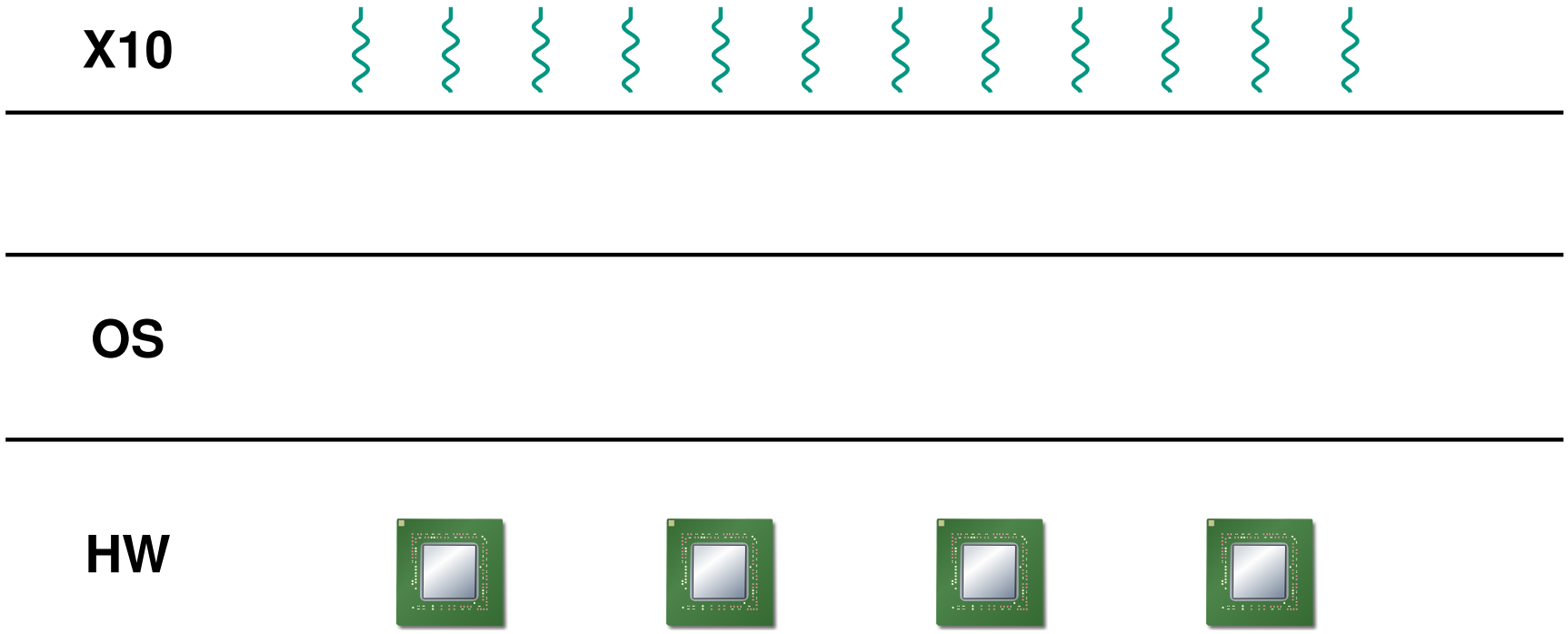
Ideal World



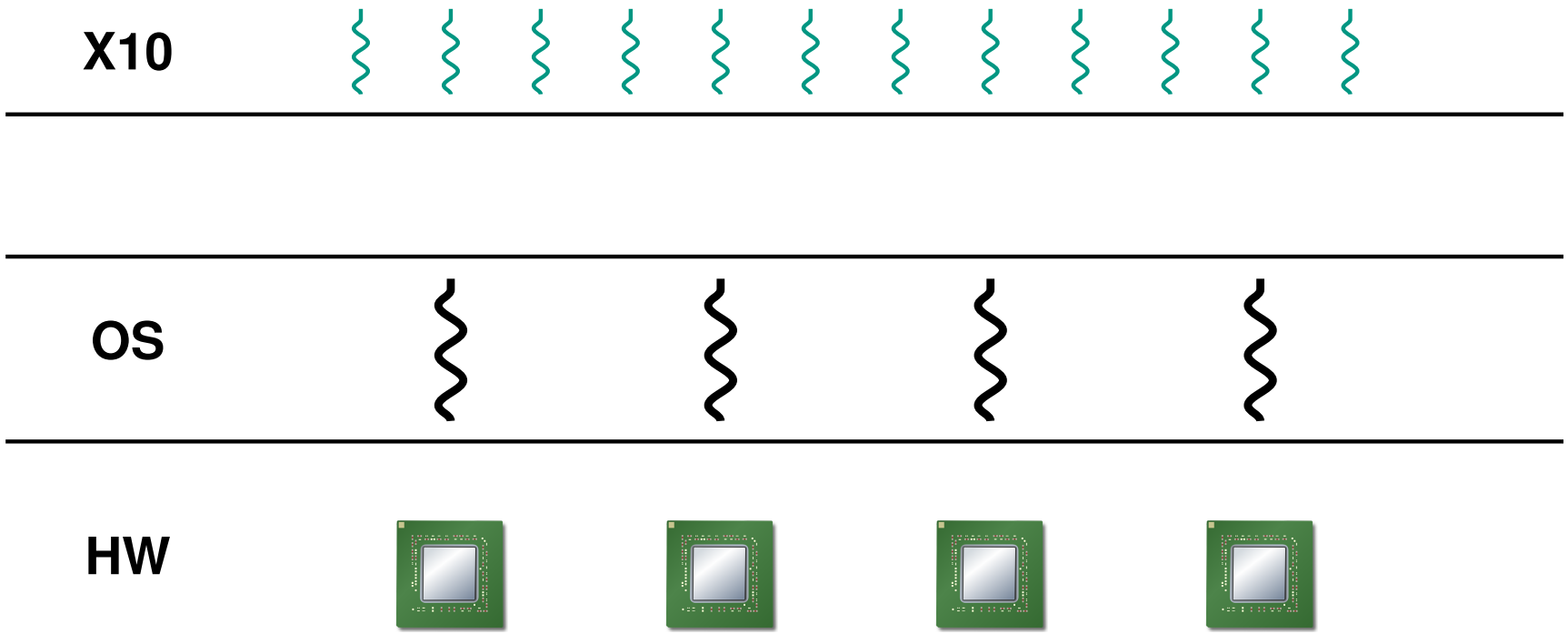
Ideal World



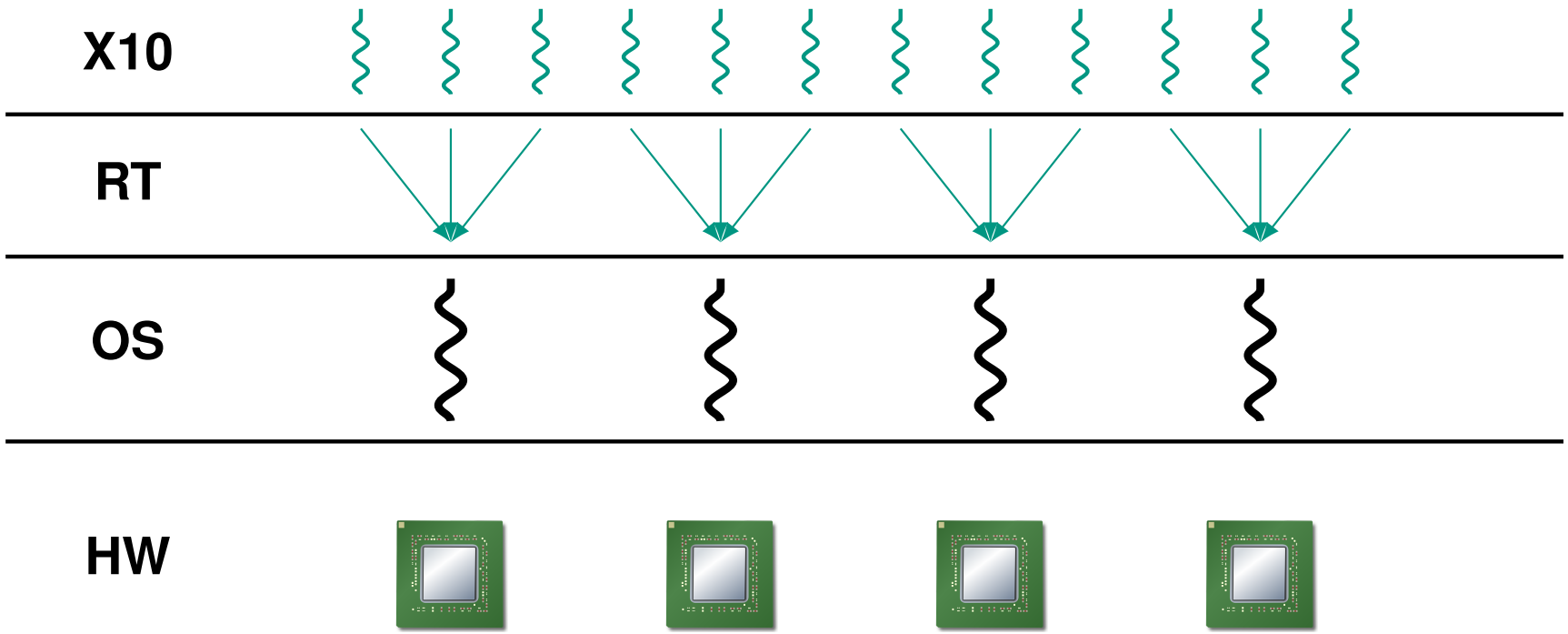
Current Practice



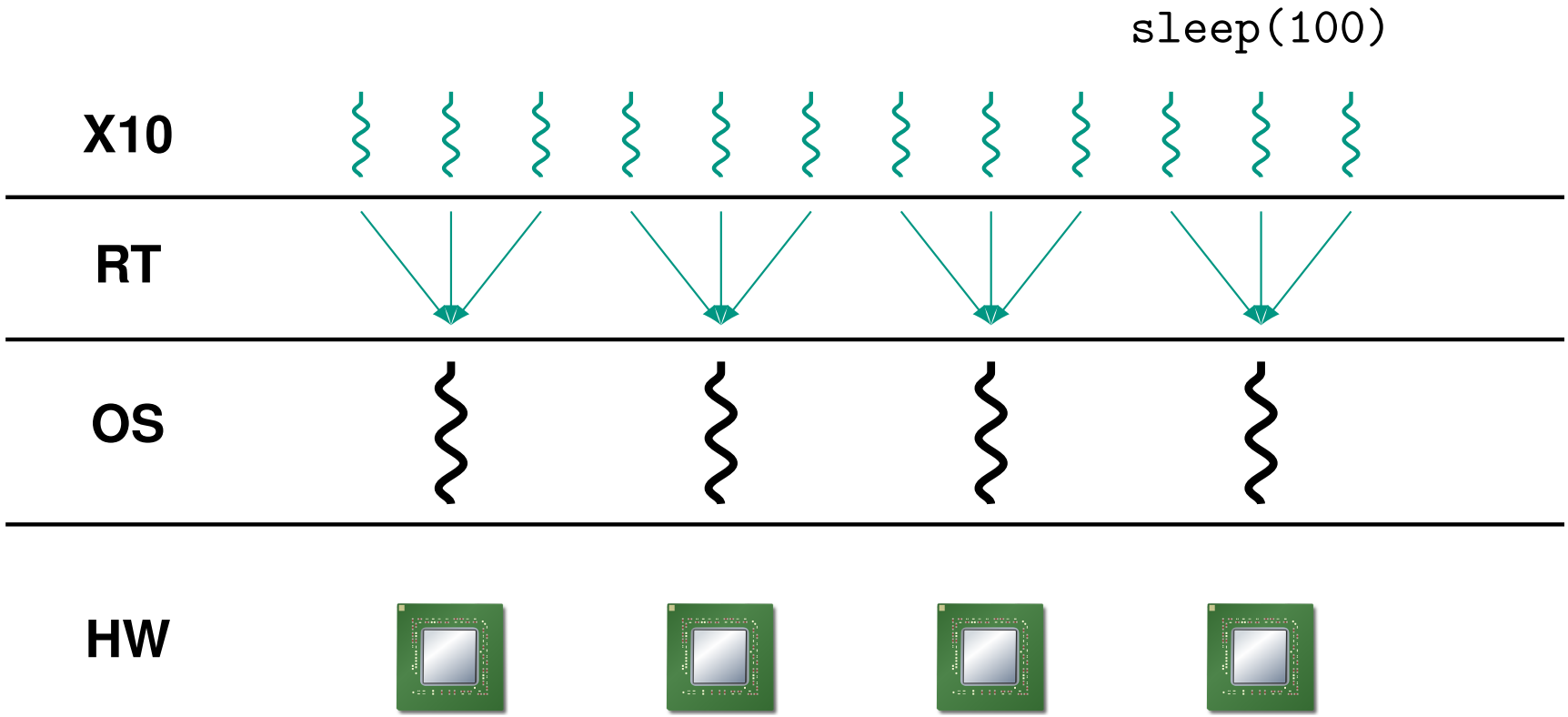
Current Practice



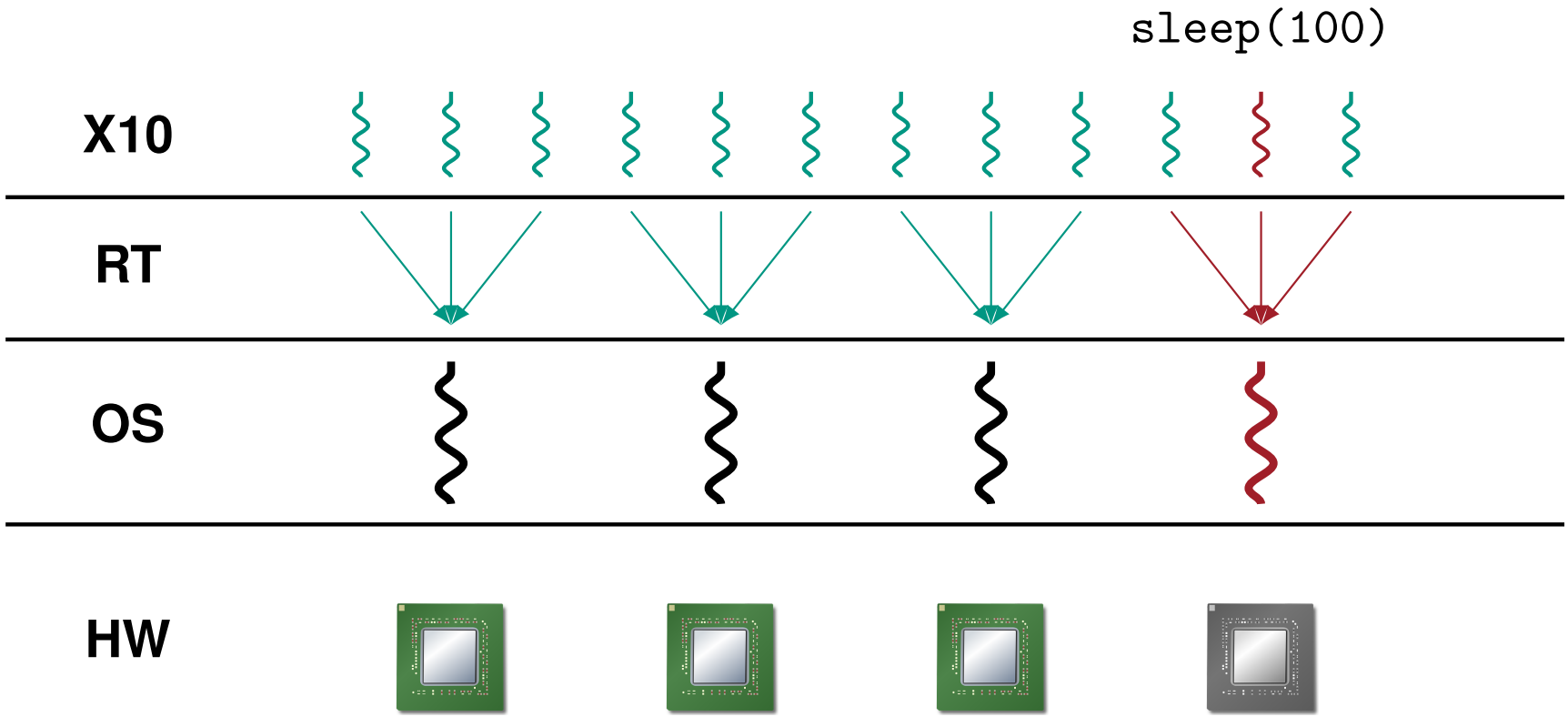
Current Practice



Current Practice



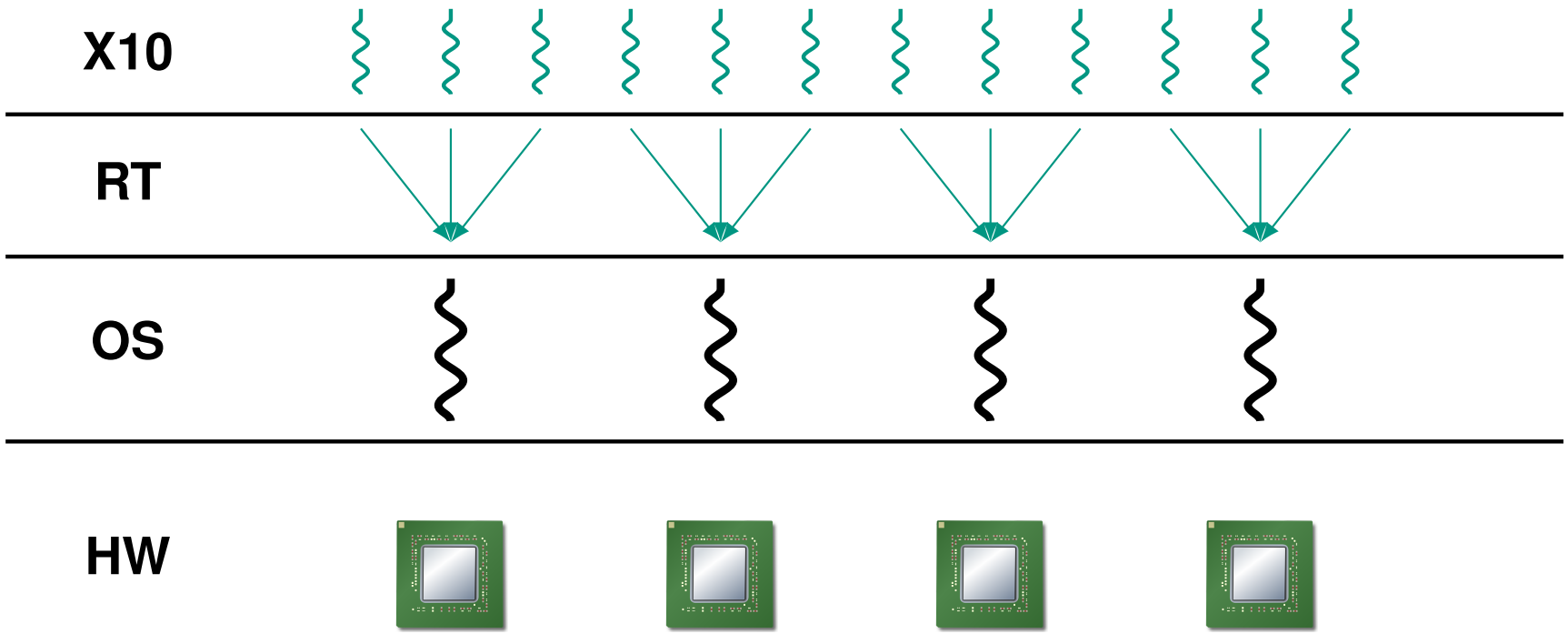
Current Practice



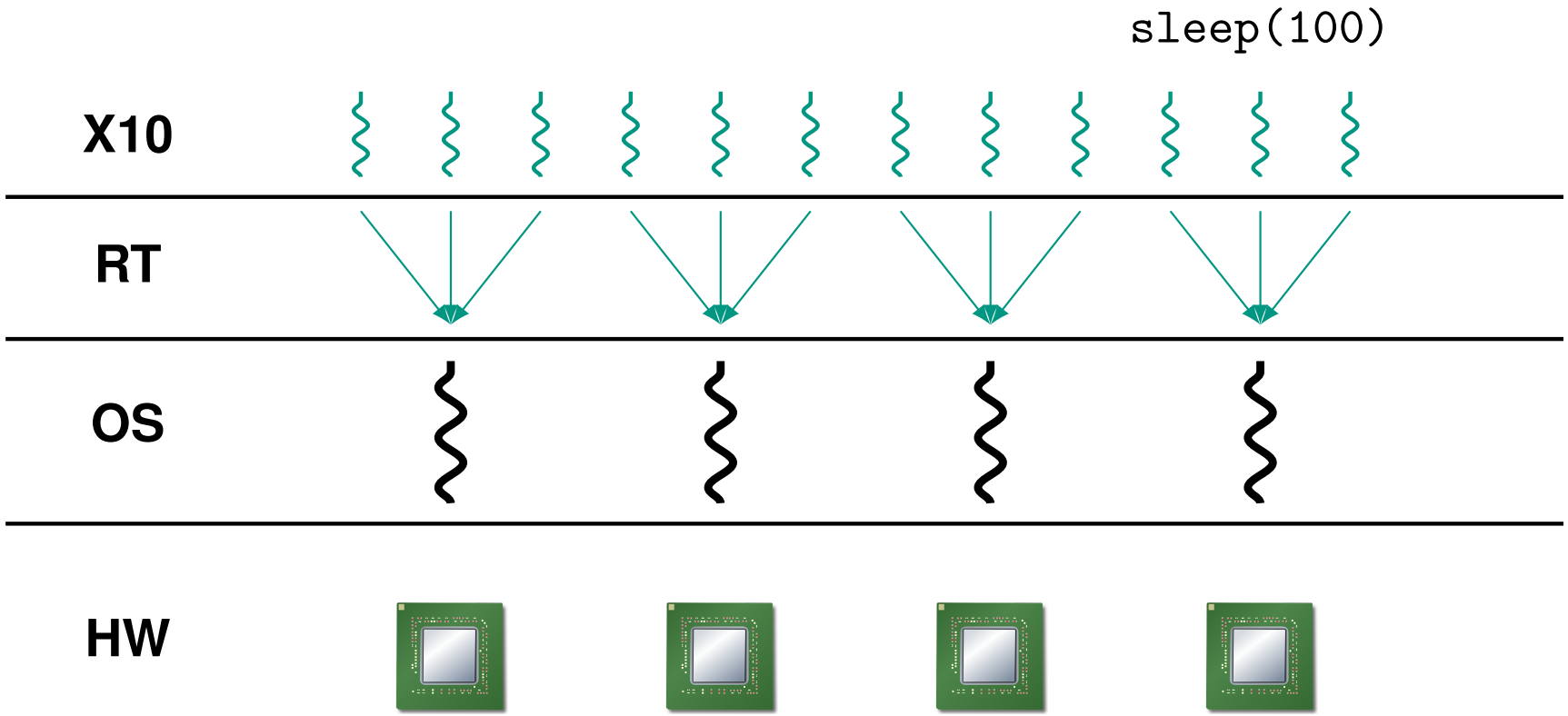
Workaround

```
class Runtime {
  public static def sleep(millis:Long) {
    Runtime.increaseParallelism();
    Thread.sleep(millis);
    Runtime.decreaseParallelism(1);
  }
}
```

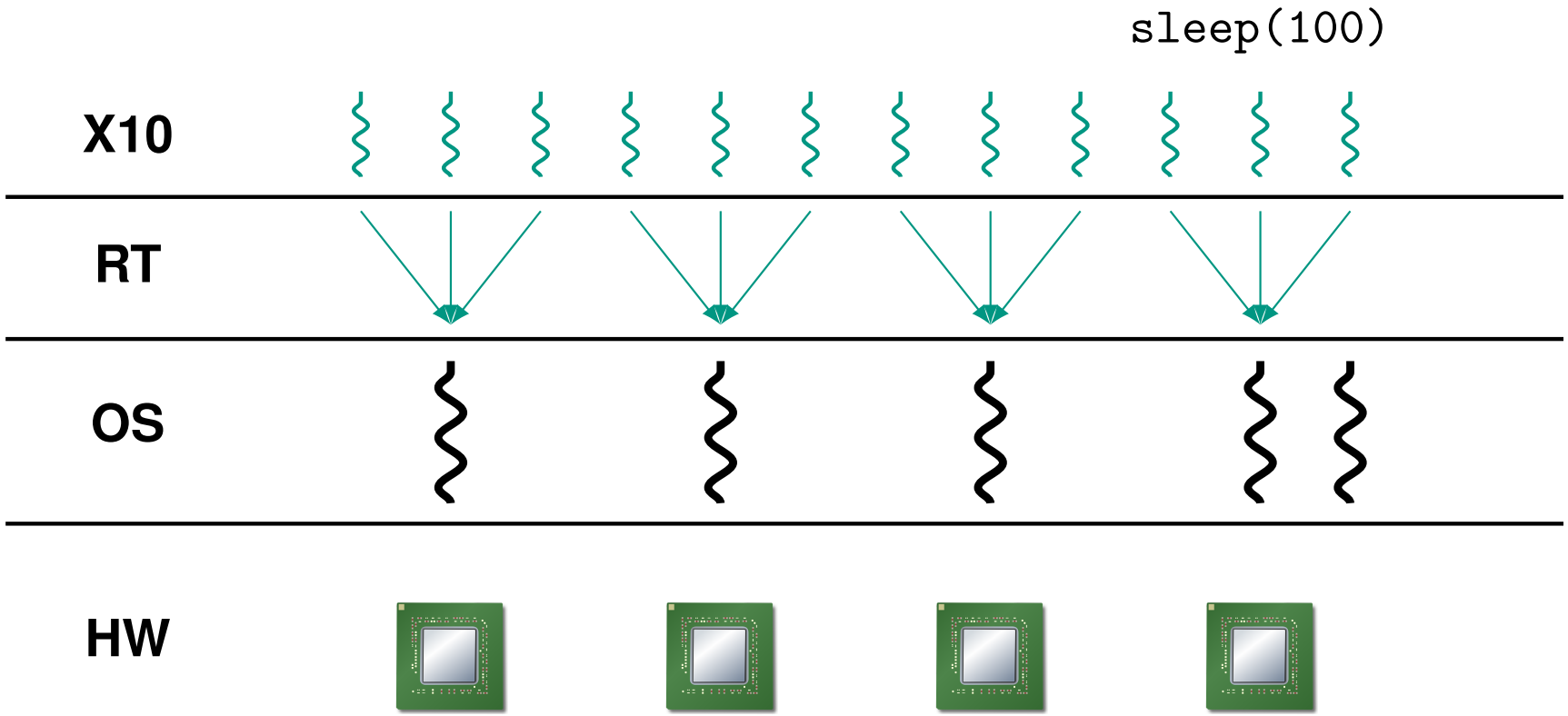
Workaround in Action



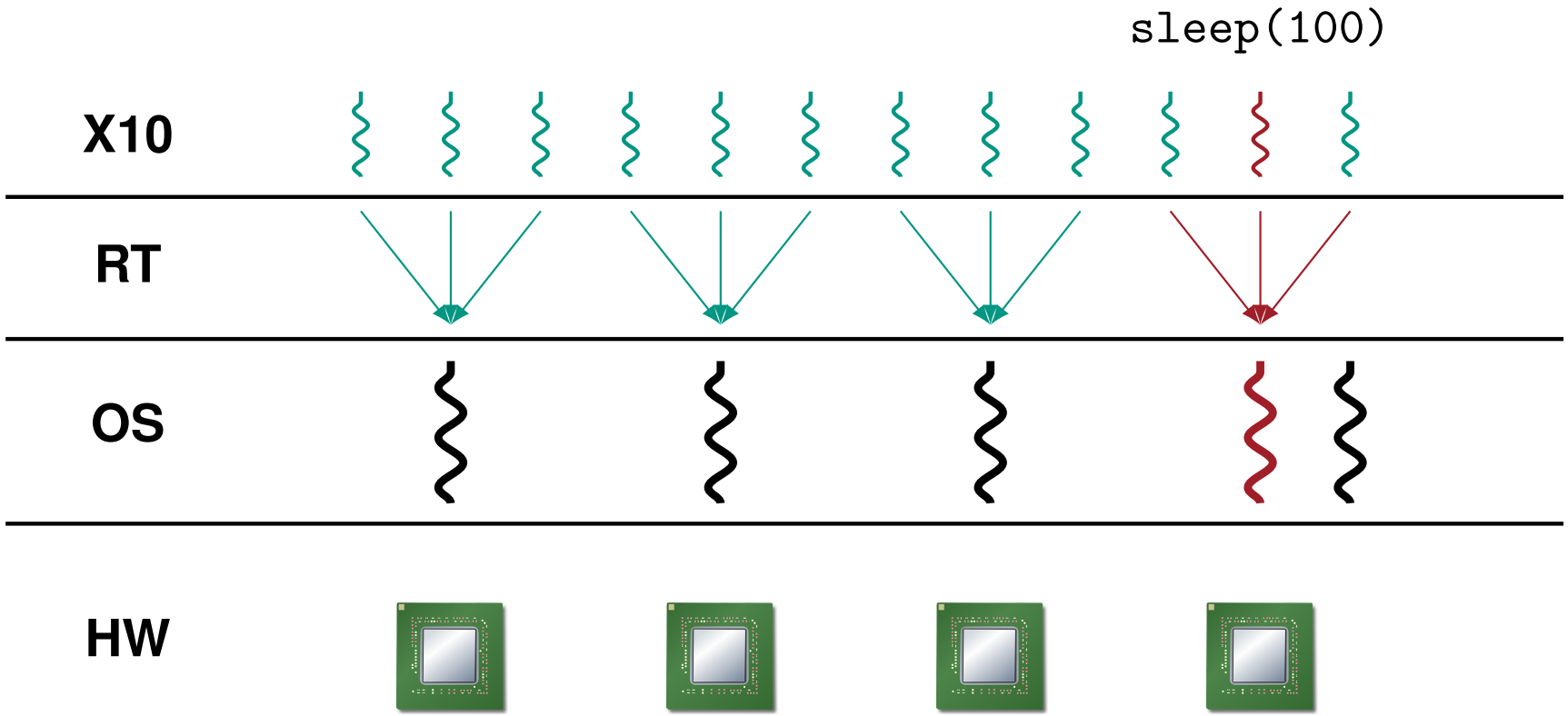
Workaround in Action



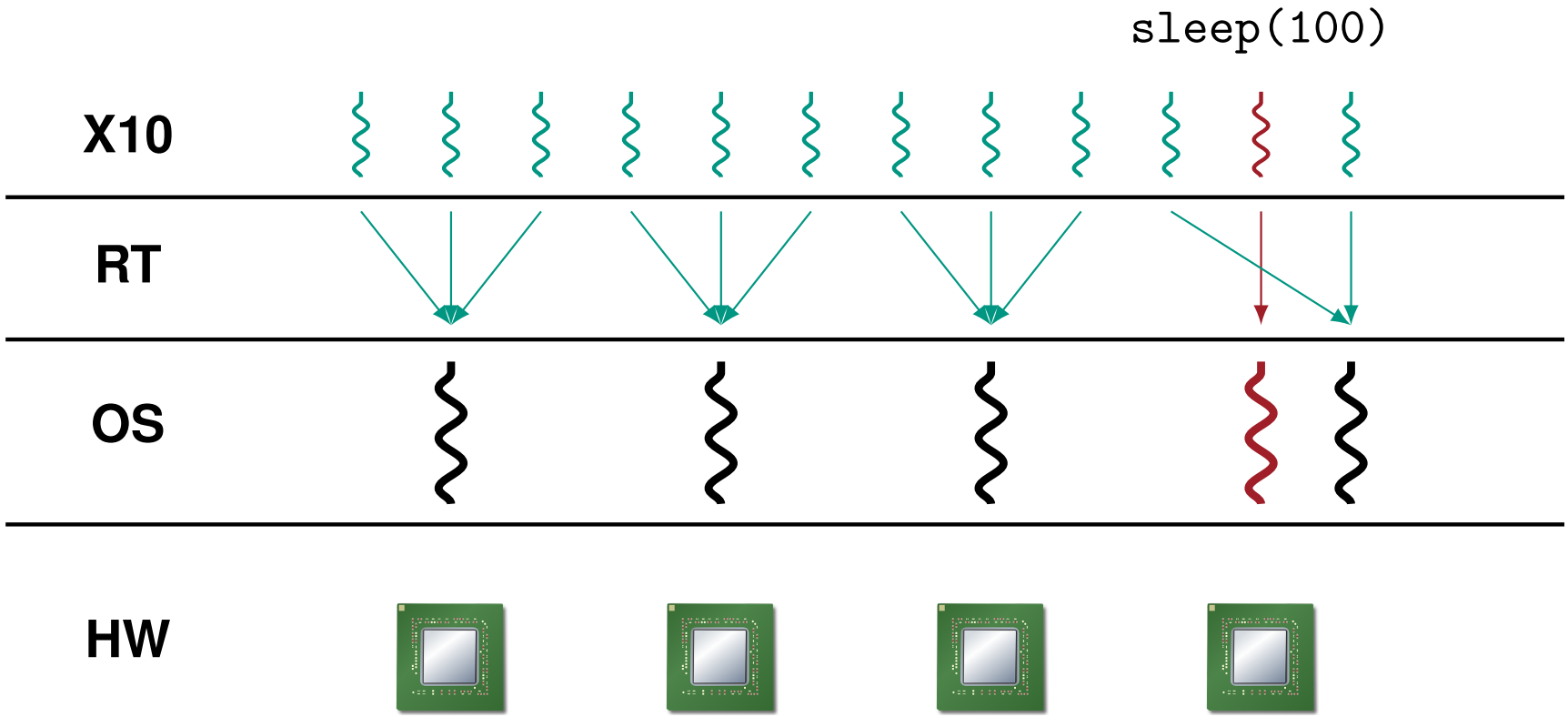
Workaround in Action



Workaround in Action



Workaround in Action



Motivation

Problems with user-level scheduling approach:

- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)

Motivation

Problems with user-level scheduling approach:

- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)

$\Rightarrow$ Why not activity = OS-level primitive?

Motivation

Problems with user-level scheduling approach:

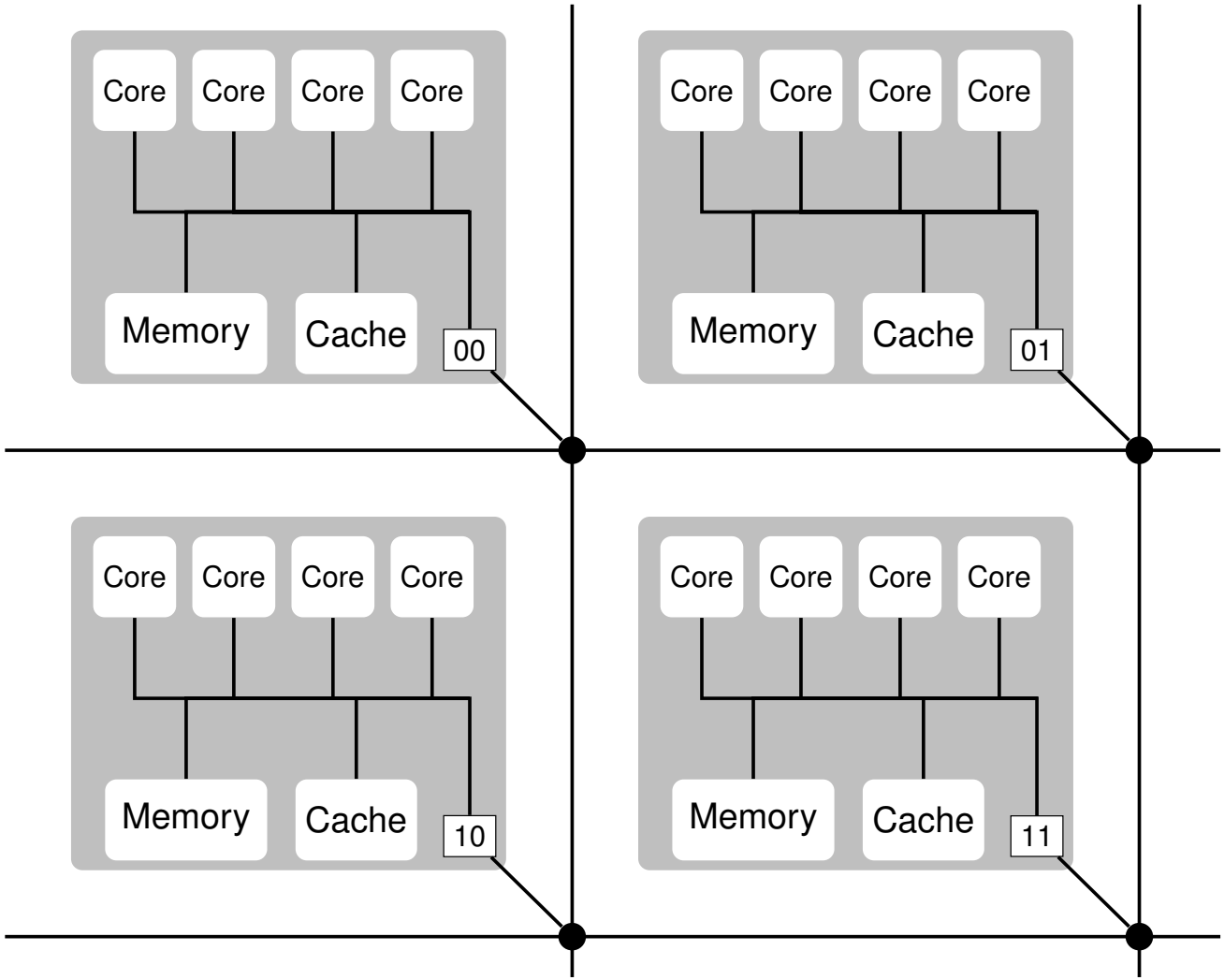
- Complexity: interplay between two schedulers
- Performance: starting/stopping kernel-level threads is expensive
- Bugs: what if starting/stopping is forgotten? (e.g., user code)

$\Rightarrow$ Why not activity = OS-level primitive?

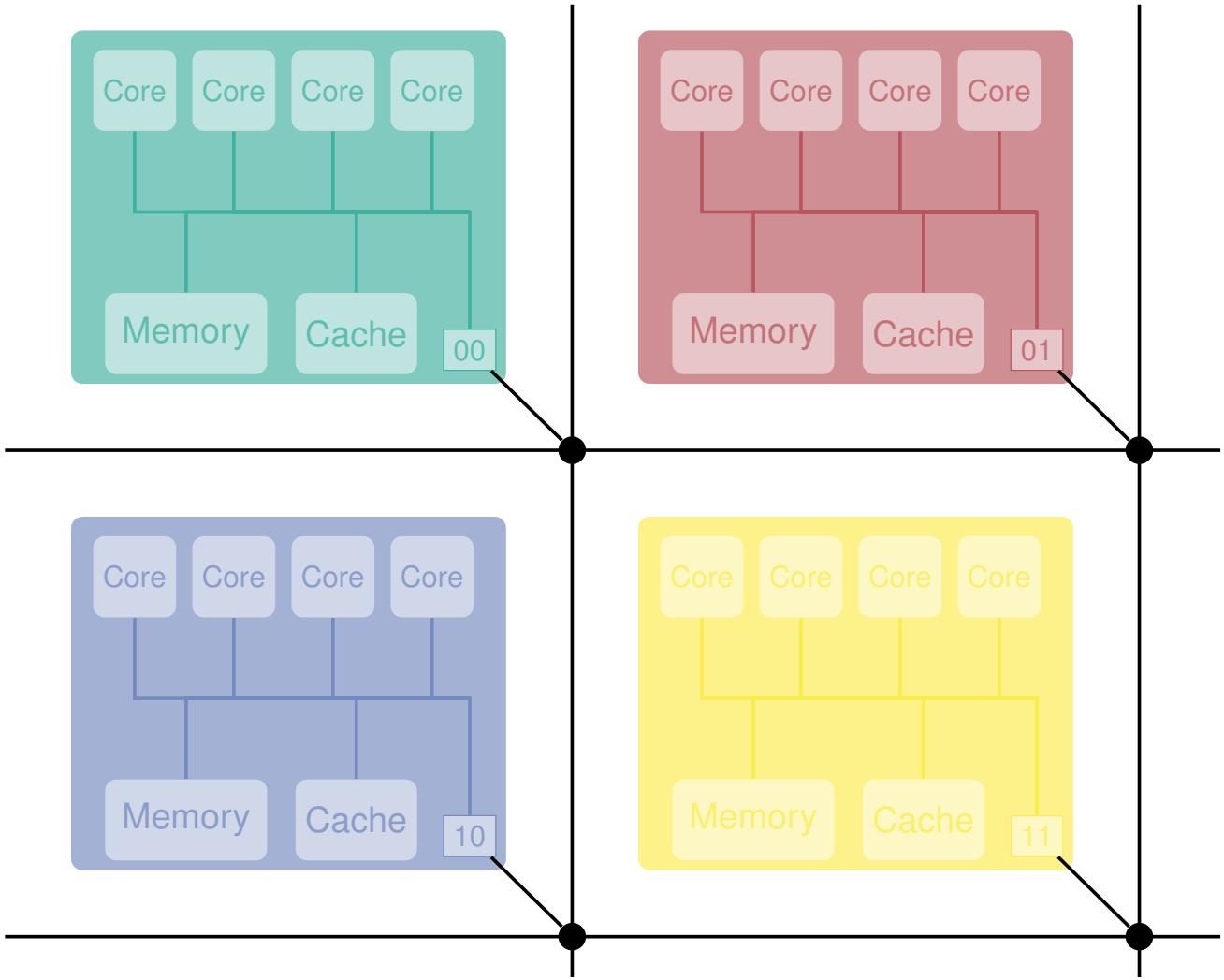
In this talk:

- How we directly mapped activities to OS primitives
  - Context: many-core hardware architecture
- How this simplifies runtime system and OS
- Initial evaluation of system efficiency

Tiled Many-Core Architectures



Tiled Many-Core Architectures



OctoPOS

- OS designed for many-core PGAS architectures

OctoPOS

- OS designed for many-core PGAS architectures

PGAS Architecture

$\Rightarrow$ One OS instance per place

$\Rightarrow$ Message passing

OctoPOS

- OS designed for many-core PGAS architectures

Many-Core

⇒ Enough cores for *exclusive* allocation

⇒ Cooperative scheduling instead of preemption

PGAS Architecture

⇒ One OS instance per place

⇒ Message passing

OctoPOS

- OS designed for many-core PGAS architectures

Many-Core

⇒ Enough cores for *exclusive* allocation

⇒ Cooperative scheduling instead of preemption

PGAS Architecture

⇒ One OS instance per place

⇒ Message passing

- User-level-like scheduler in the kernel
  - Cooperative FIFO scheduling

OctoPOS

- OS designed for many-core PGAS architectures

Many-Core

⇒ Enough cores for *exclusive* allocation

⇒ Cooperative scheduling instead of preemption

PGAS Architecture

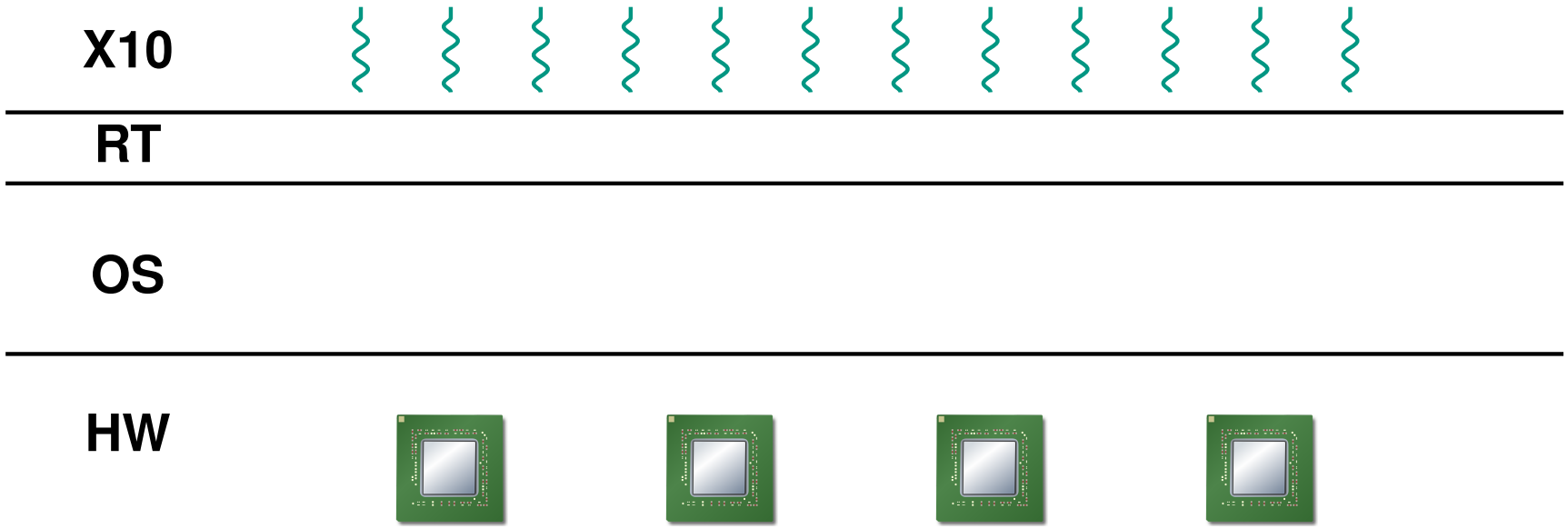
⇒ One OS instance per place

⇒ Message passing

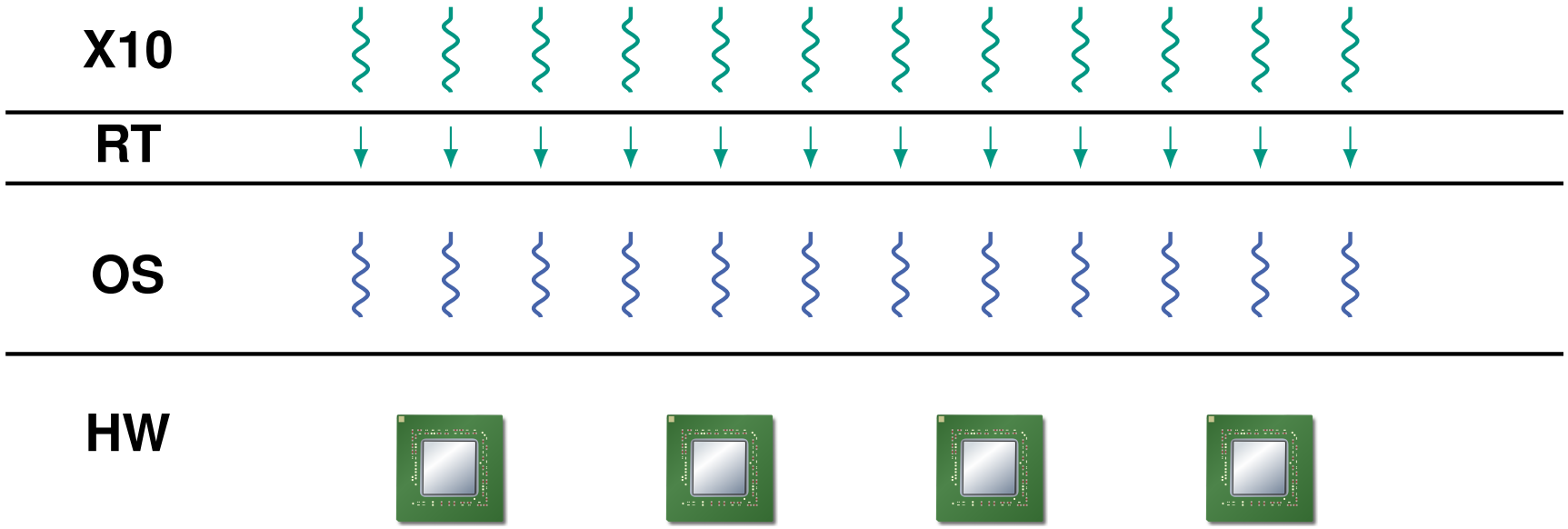
- User-level-like scheduler in the kernel
  - Cooperative FIFO scheduling

⇒ Very lightweight threads called ***i*-lets**

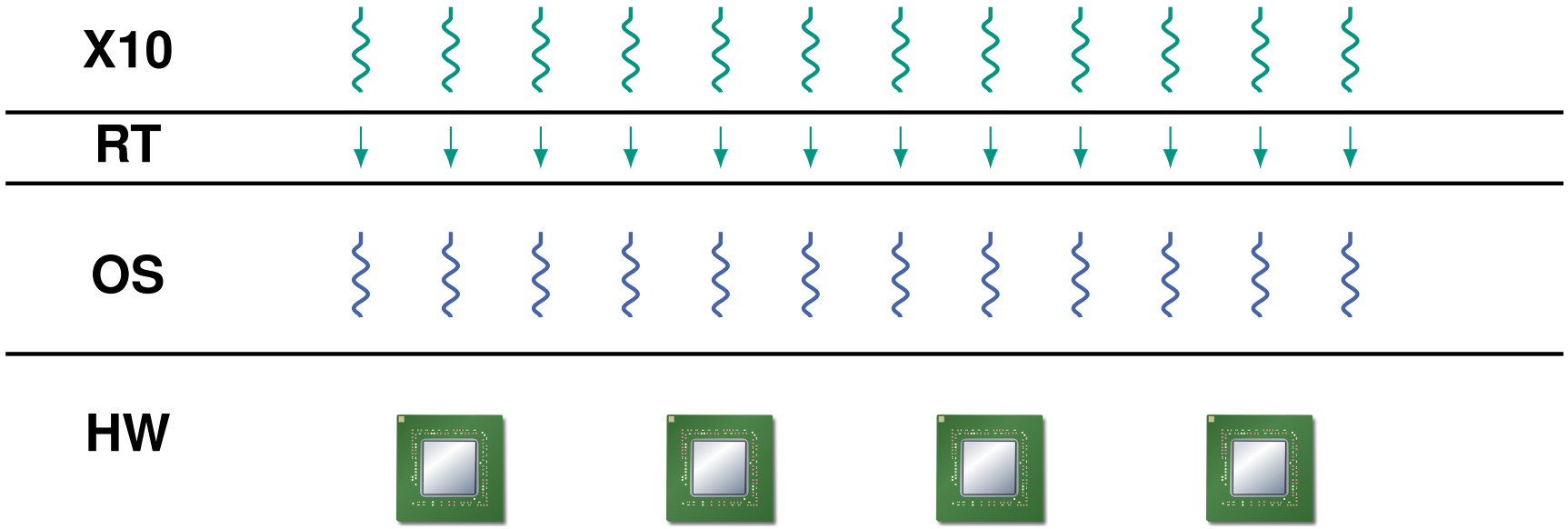
Async



Async



Async



- Each activity corresponds to exactly one *i*-let
- Very thin runtime system, no user-level scheduler
- Blocking calls unproblematic, no workaround needed

Remote *i*-let spawning

Karlsruhe Institute of Technology

```
spawn_ilet(place_id, ilet)
```

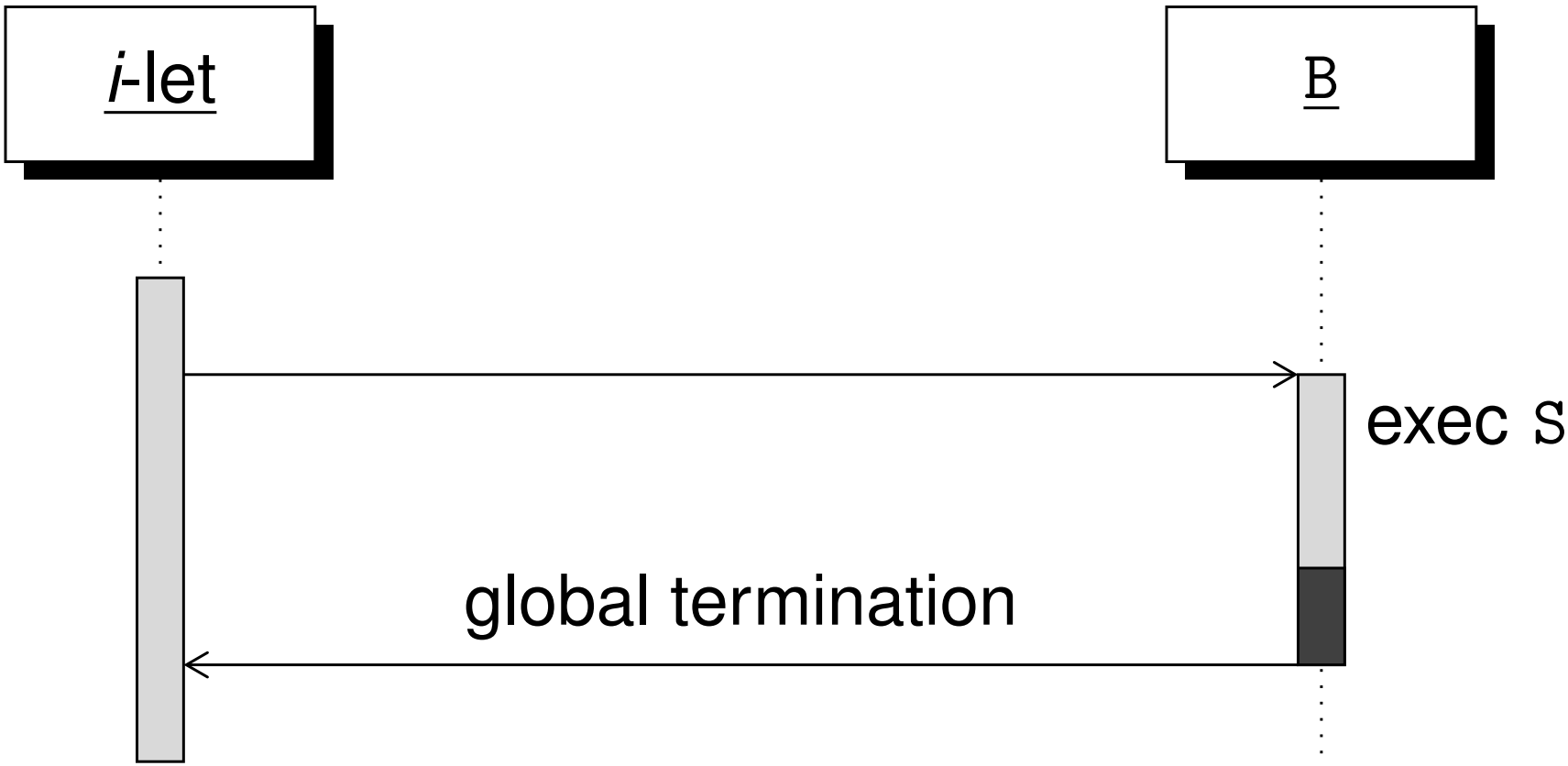
- Start an *i*-let on a different place
- Asynchronous

Small At Async Statement: `at (B) async S`

Small At Async Statement: `at (B) async S`

Small At Async Statement: `at (B) async S`

Small At Async Statement: `at (B) async S`

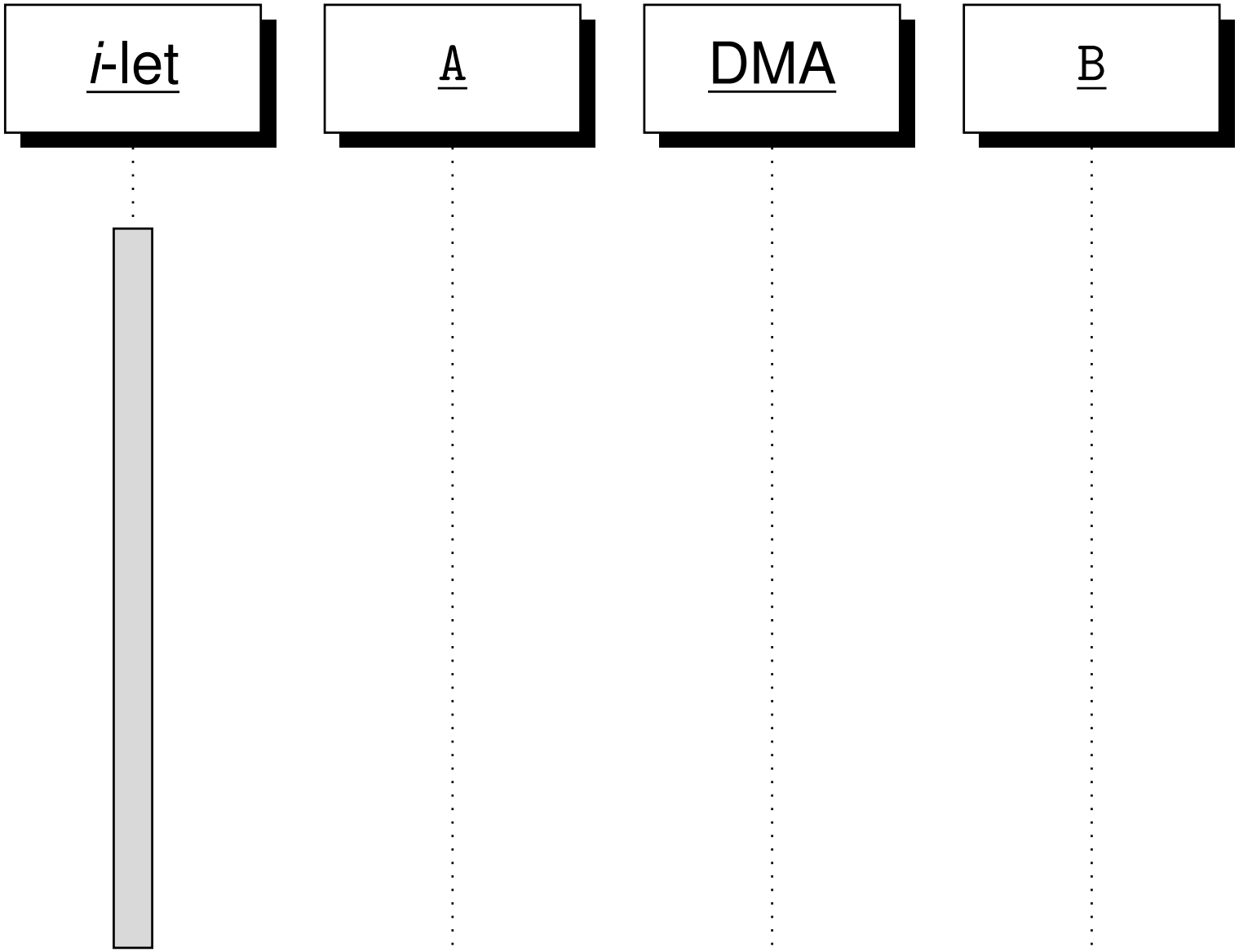


Push DMA Transfer

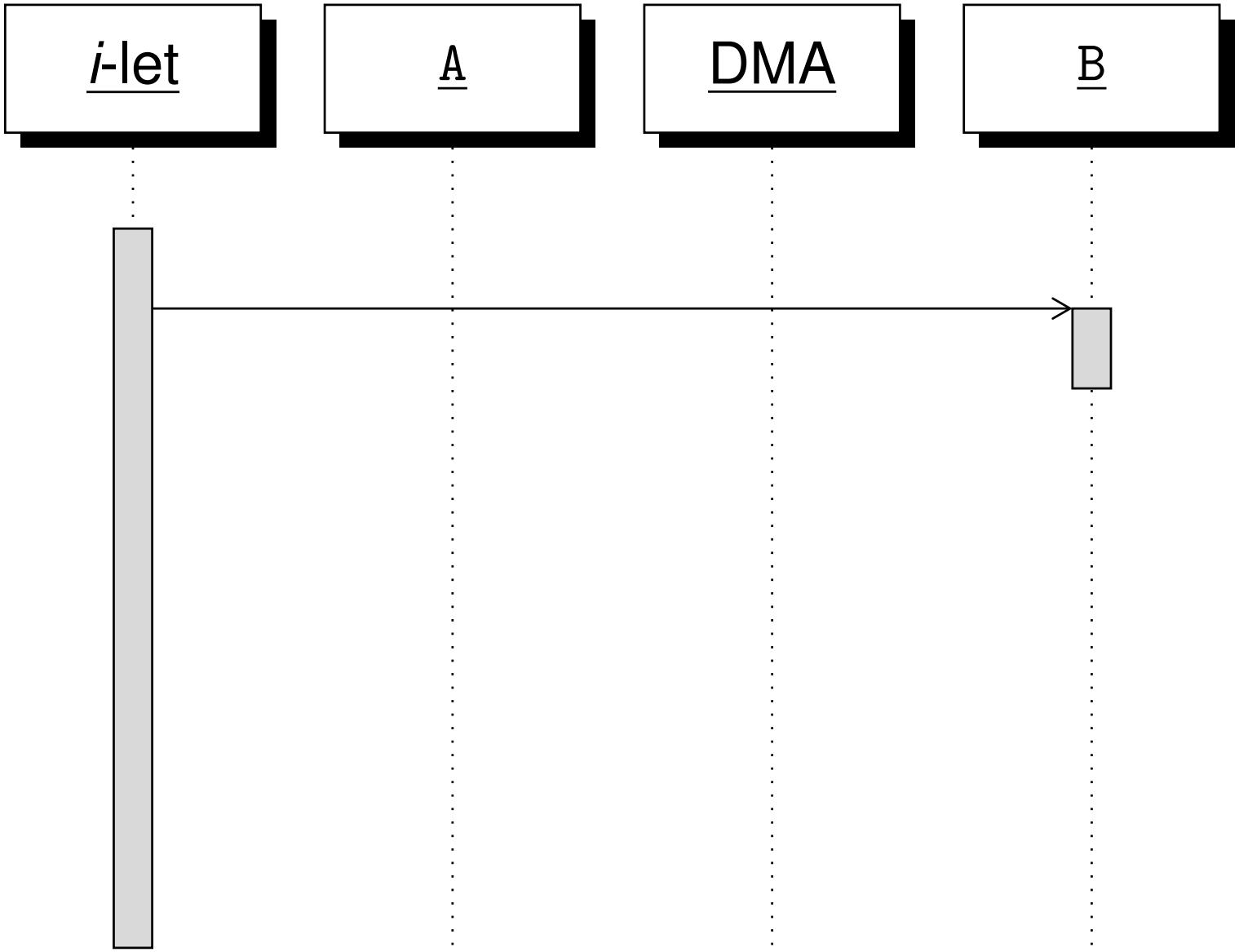
```
push_dma(place_id, data, length, sender_ilet, receiver_ilet)
```

- Copy memory block to different place
- Specify actions to be executed when transfer is finished
- Asynchronous, HW support

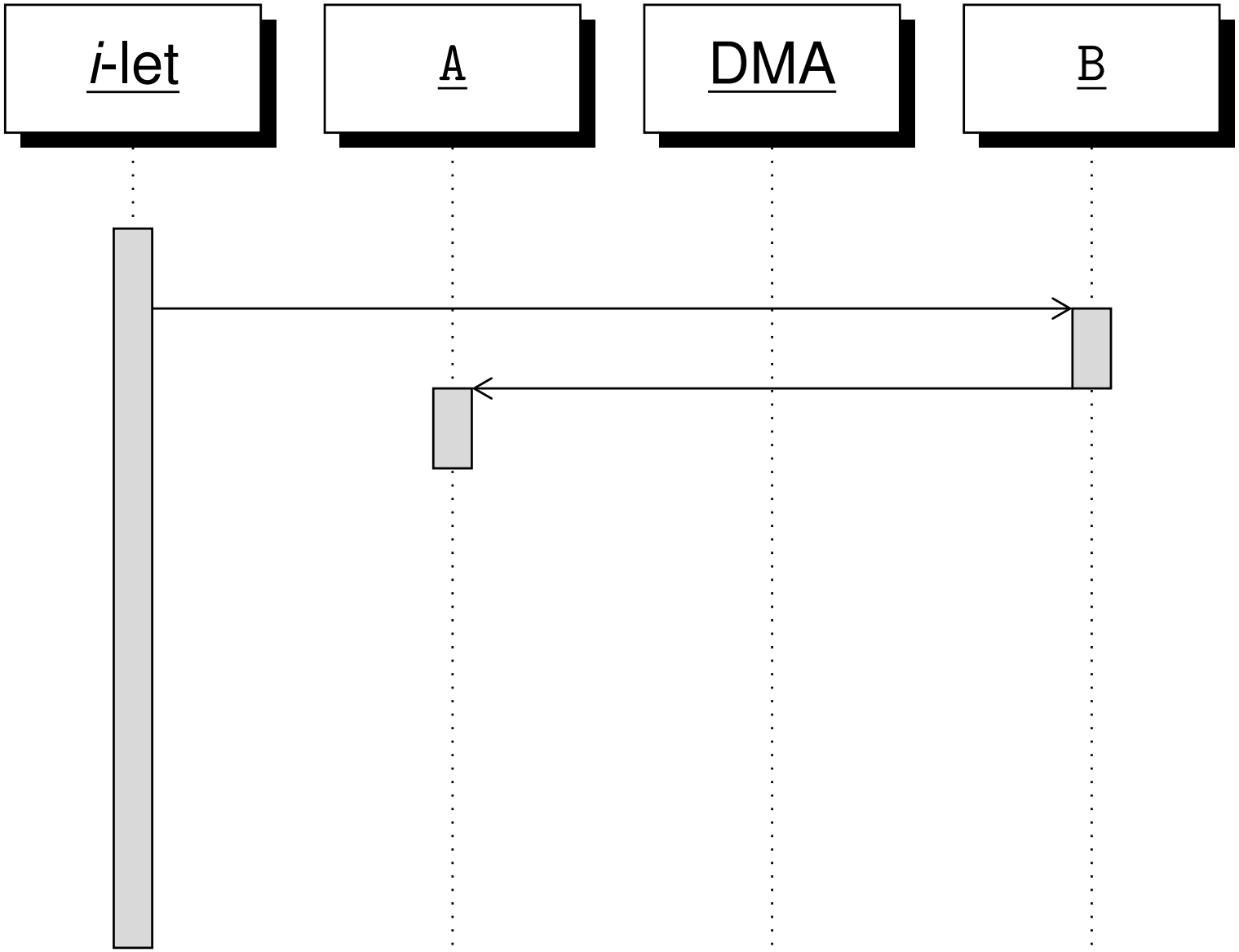
At Async Statement: `at (B) async S`



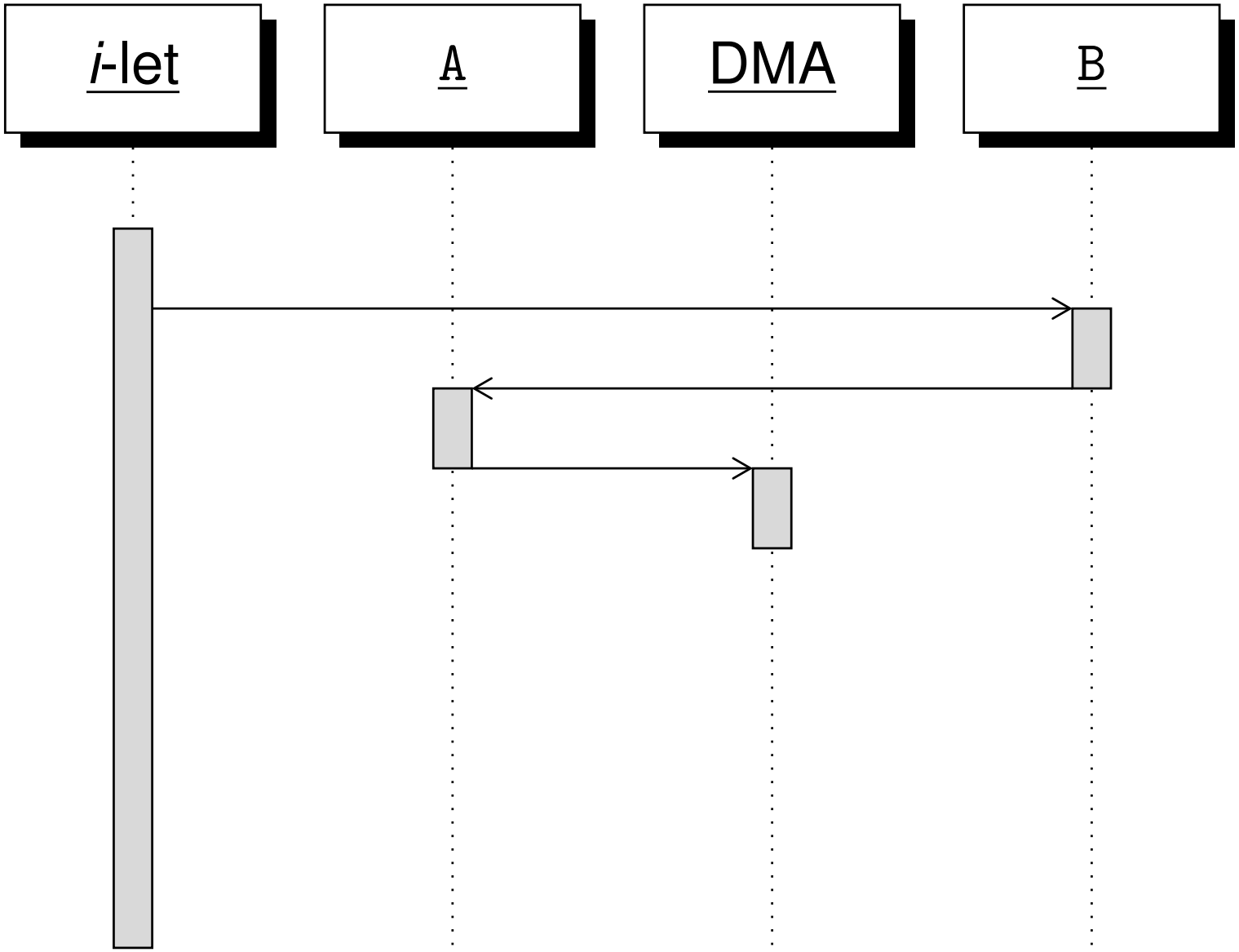
At Async Statement: `at (B) async S`



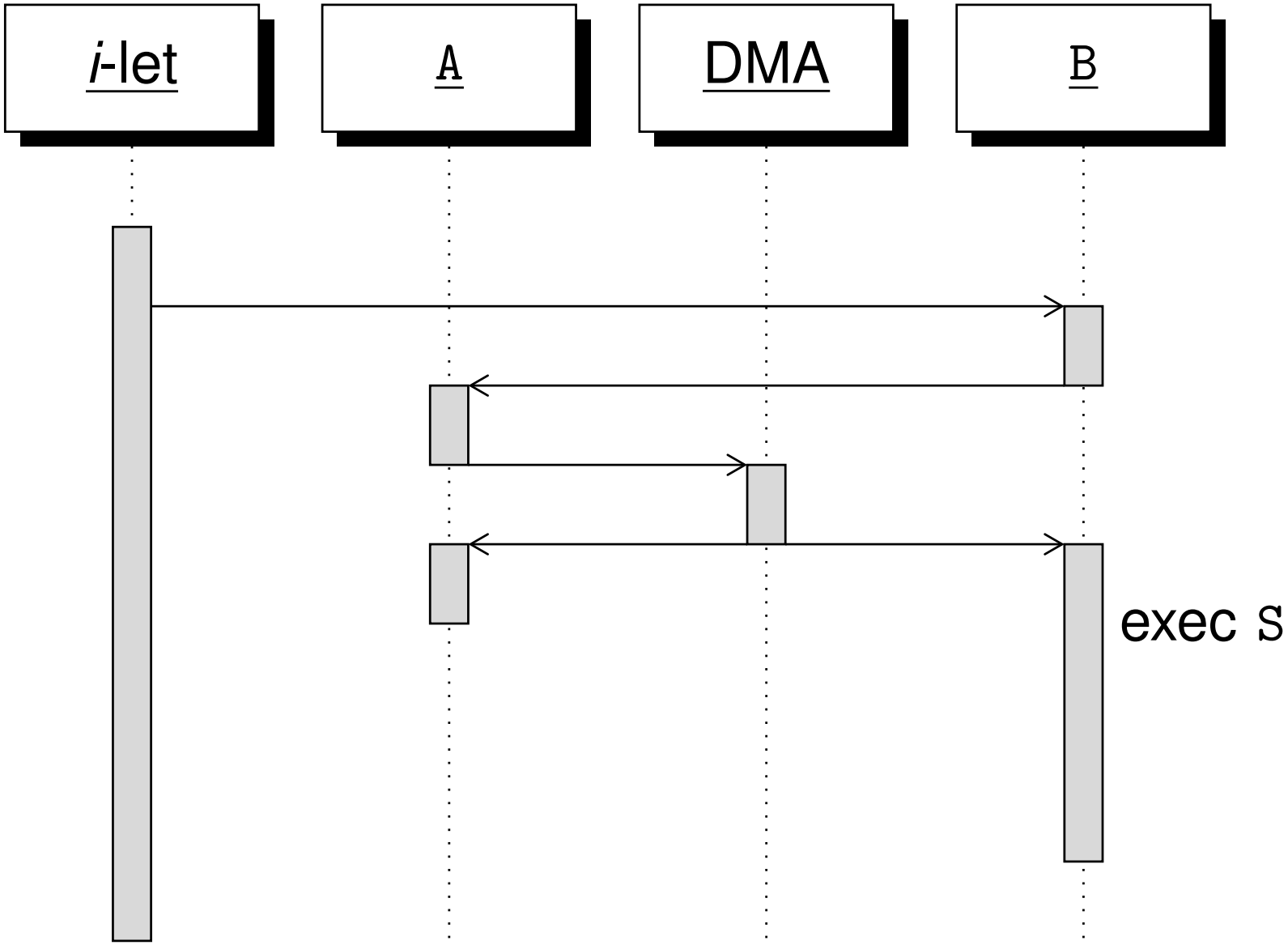
At Async Statement: `at (B) async S`



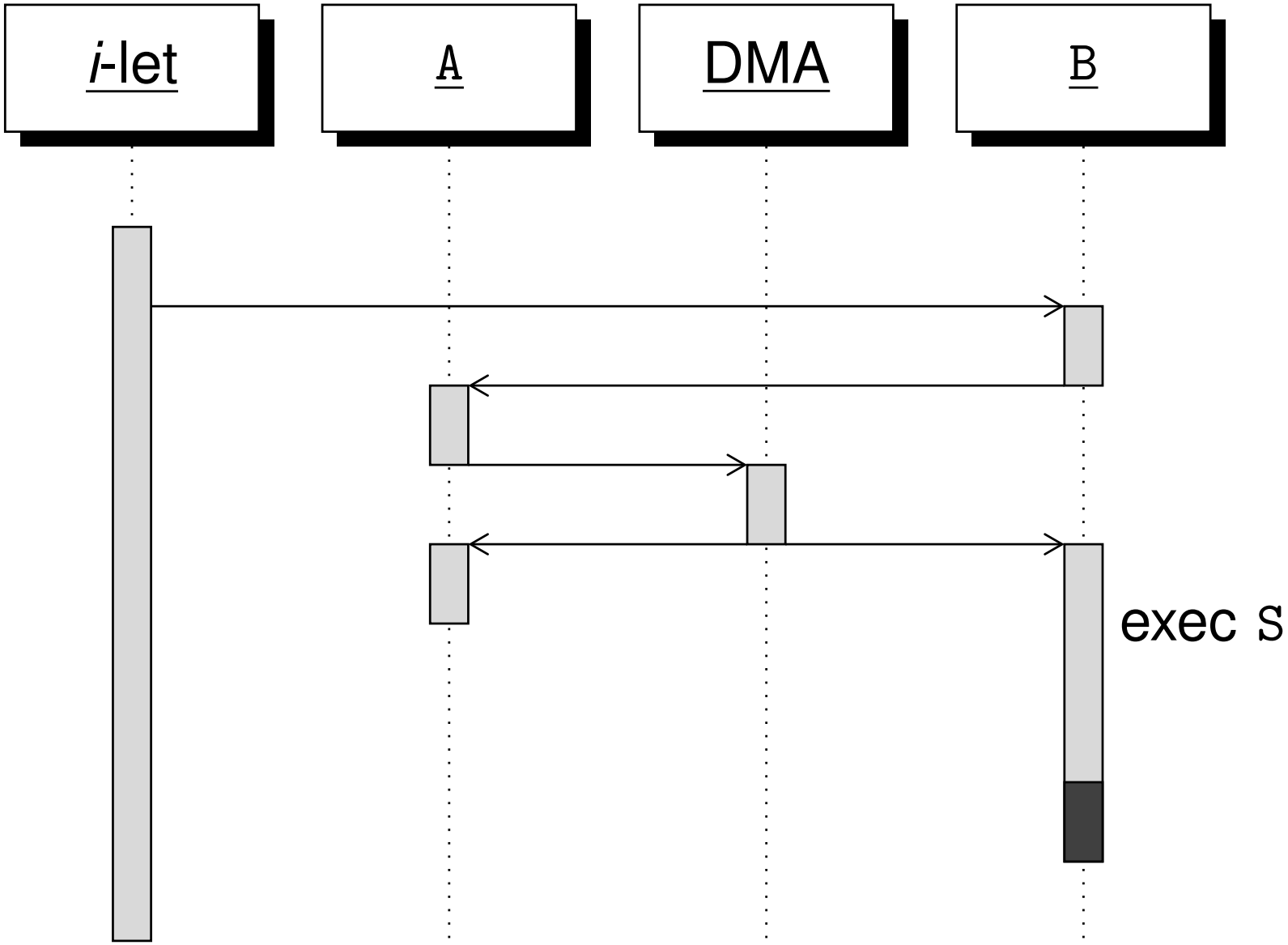
At Async Statement: `at (B) async S`



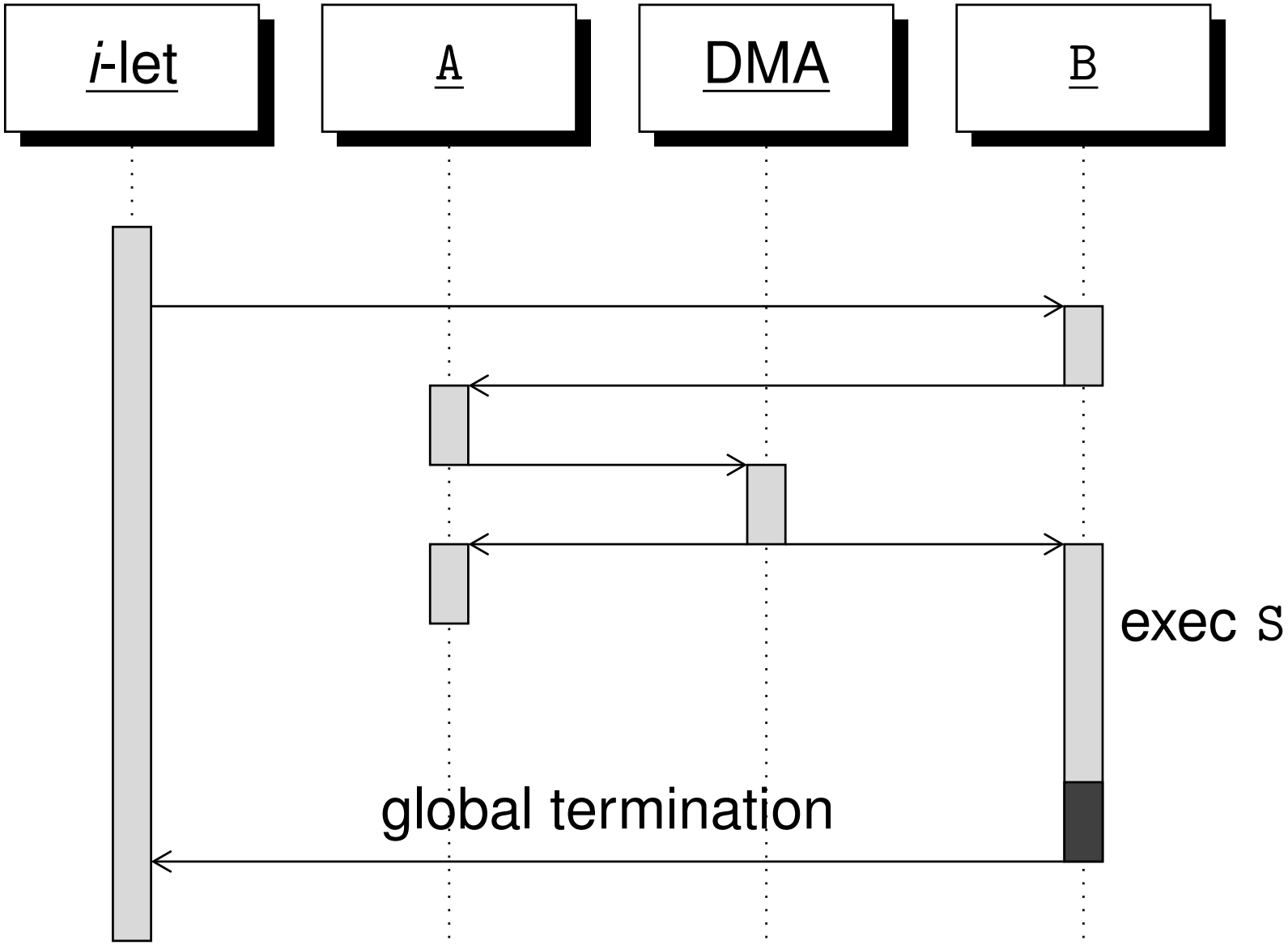
At Async Statement: `at (B) async S`



At Async Statement: `at (B) async S`



At Async Statement: `at (B) async S`



Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz

Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz
- Measurements (in clock cyles):

`spawn_ilet(0, ilet)`	`async { }`
`spawn_ilet(1, ilet)`	`at (Place(1)) async { }`

Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz
- Measurements (in clock cyles):

`spawn_ilet(0, ilet)`	`async { }`
539	1469
`spawn_ilet(1, ilet)`	`at (Place(1)) async { }`
1133	1981

Evaluation

- Benchmarks on FPGA-based prototype hardware
  - 4 places with 4 cores each
  - 25 MHz
- Measurements (in clock cyles):

`spawn_ilet(0, ilet)` 539	`async { }` 1469
`spawn_ilet(1, ilet)` 1133	`at (Place(1)) async { }` 1981

- Operations are cheap (in absolute numbers)

Conclusion & Future Work

We have:

- Implemented X10 activity management without a user-level scheduler
  - Possible by exclusively allocating cores and using cooperative scheduling
  - Essentially puts user-level-like scheduler into kernel
- Adapted the X10 runtime
- Evaluated the efficiency on a prototype many-core architecture

Conclusion & Future Work

We have:

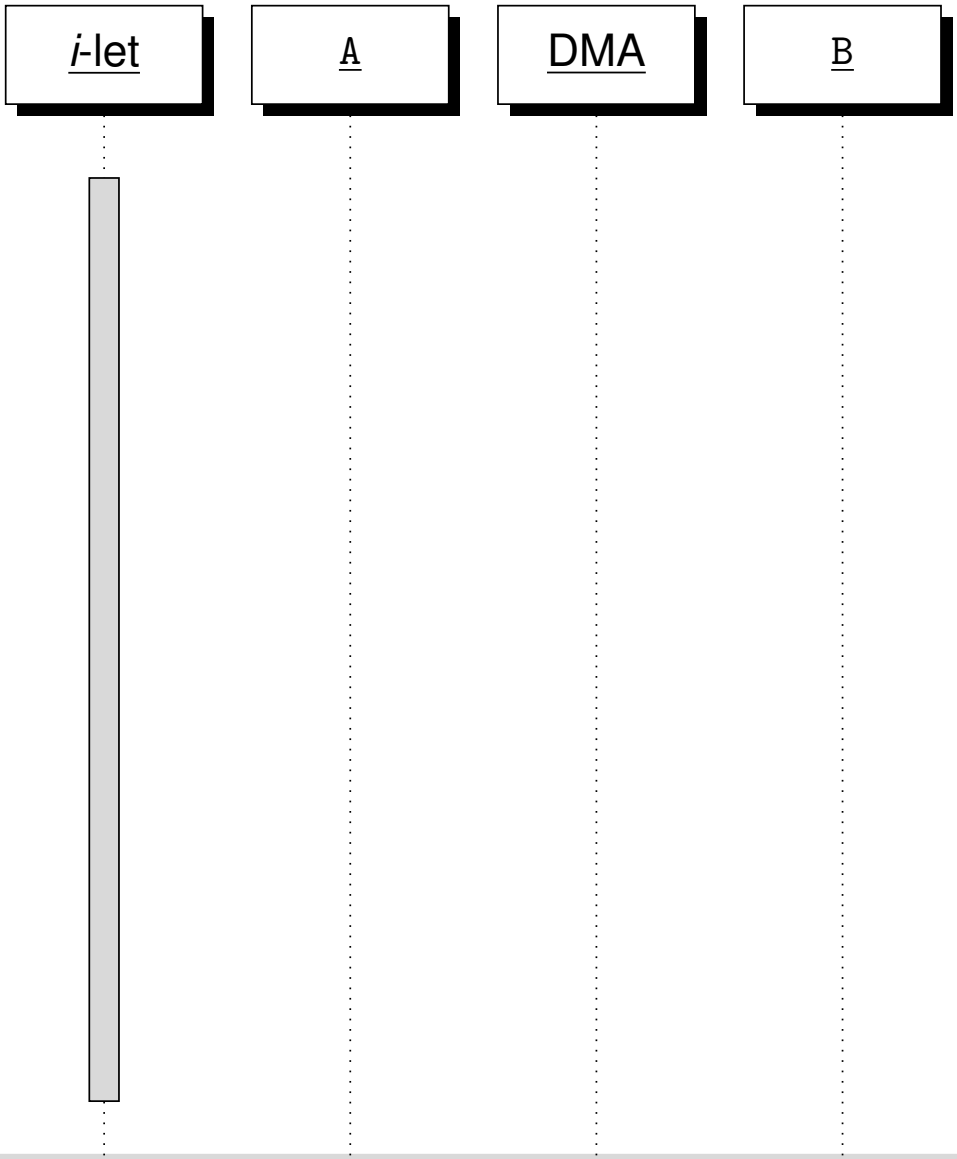
- Implemented X10 activity management without a user-level scheduler
  - Possible by exclusively allocating cores and using cooperative scheduling
  - Essentially puts user-level-like scheduler into kernel
- Adapted the X10 runtime
- Evaluated the efficiency on a prototype many-core architecture

We plan to:

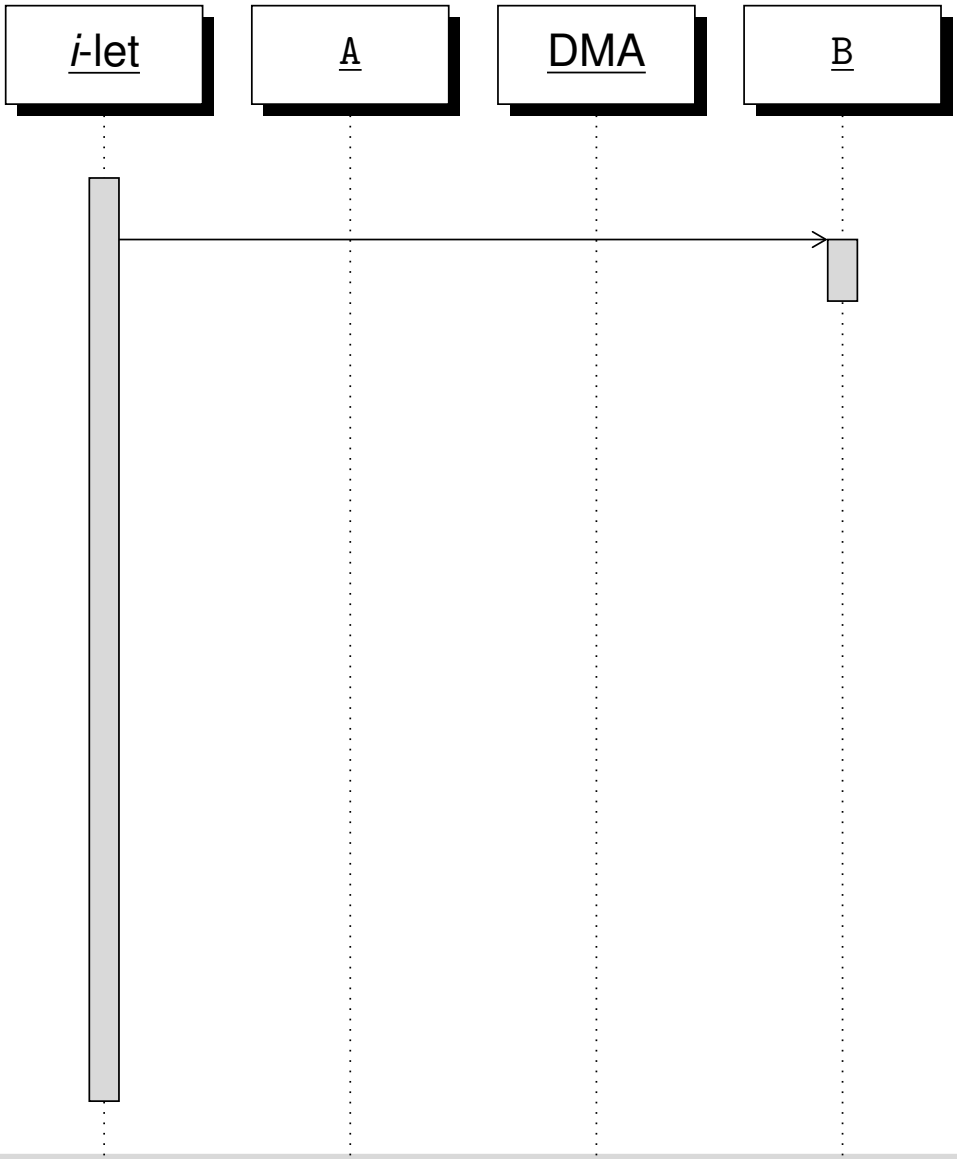
- Port OctoPOS to AMD64 NUMA systems (in progress)
- Evaluate against common Linux-MPI implementations

Backup Slides

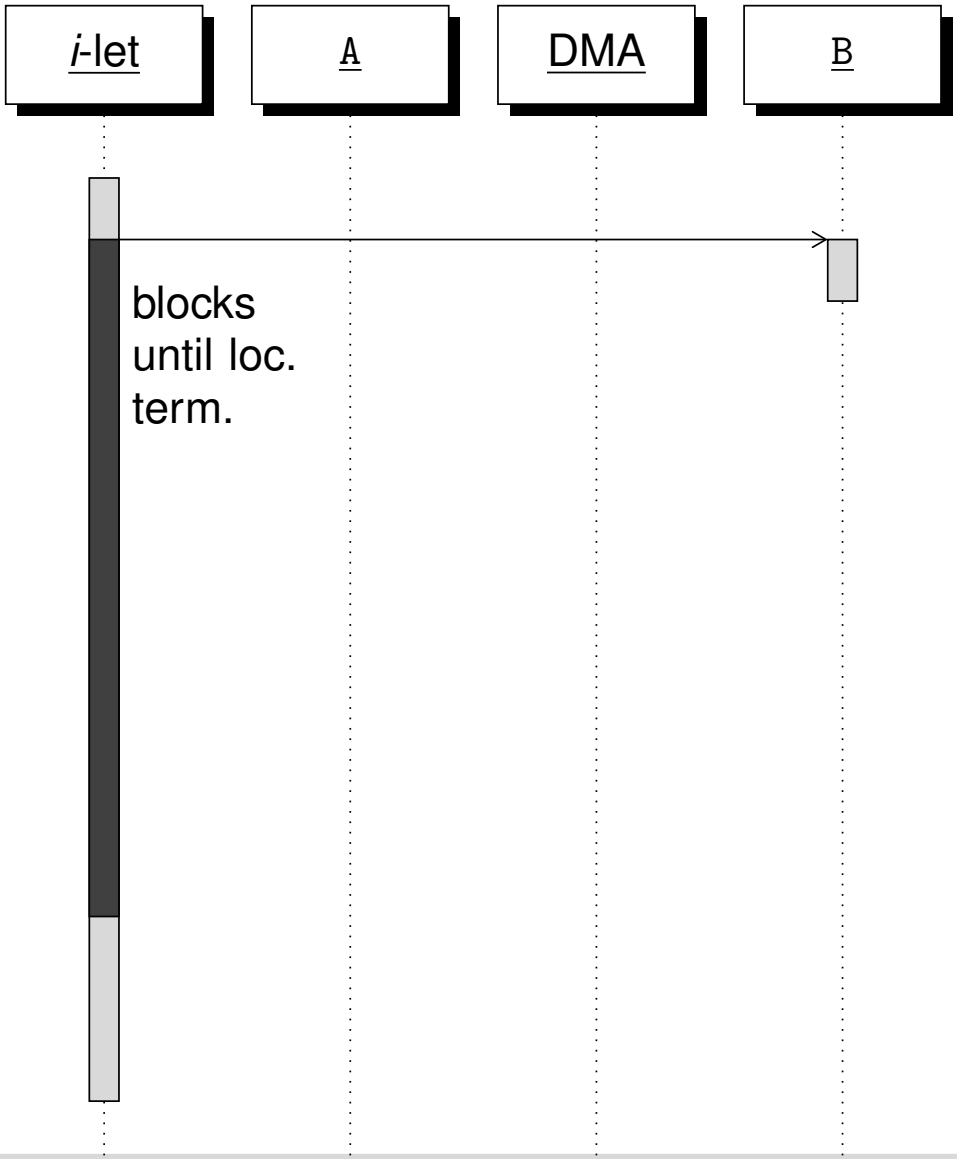
At Expression: `at (B) E`



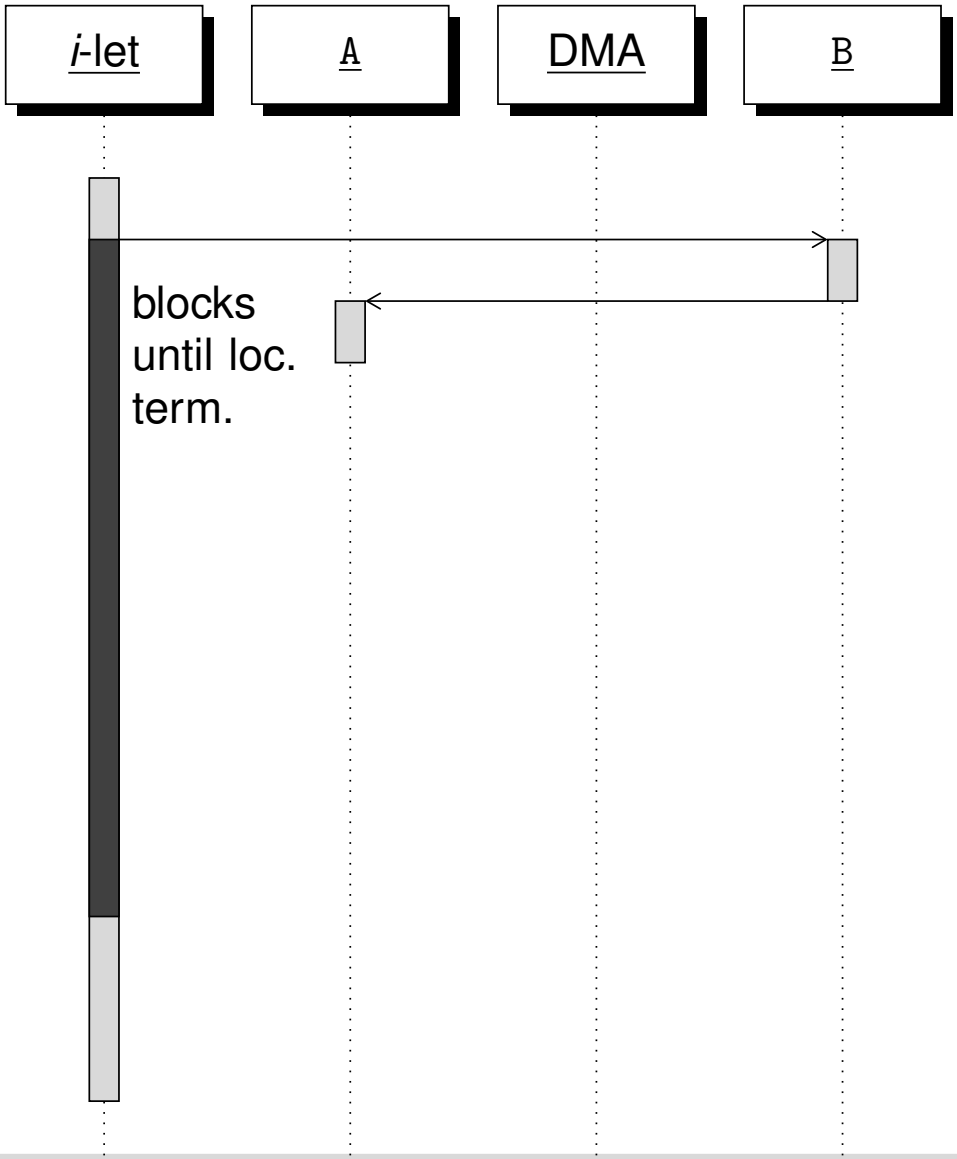
At Expression: `at (B) E`



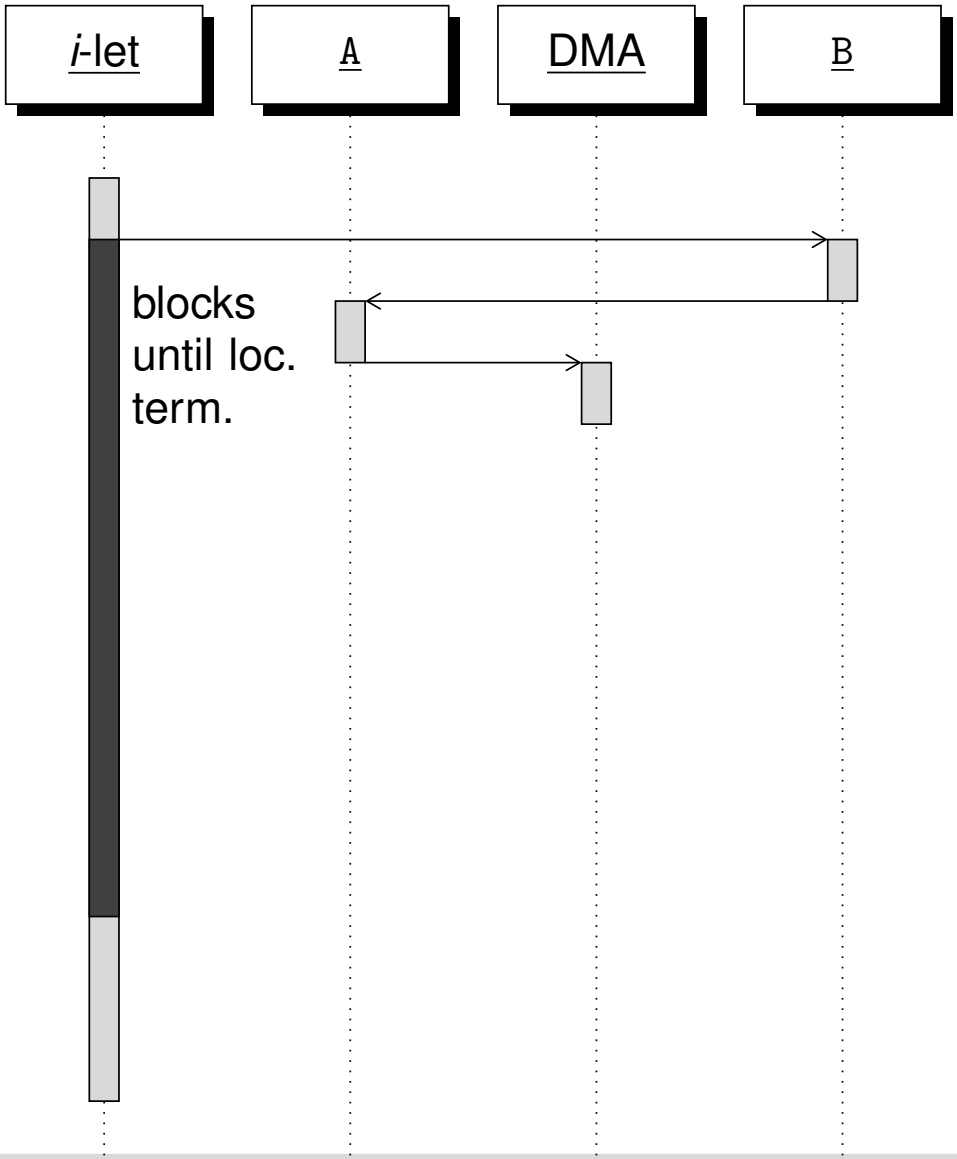
At Expression: `at (B) E`



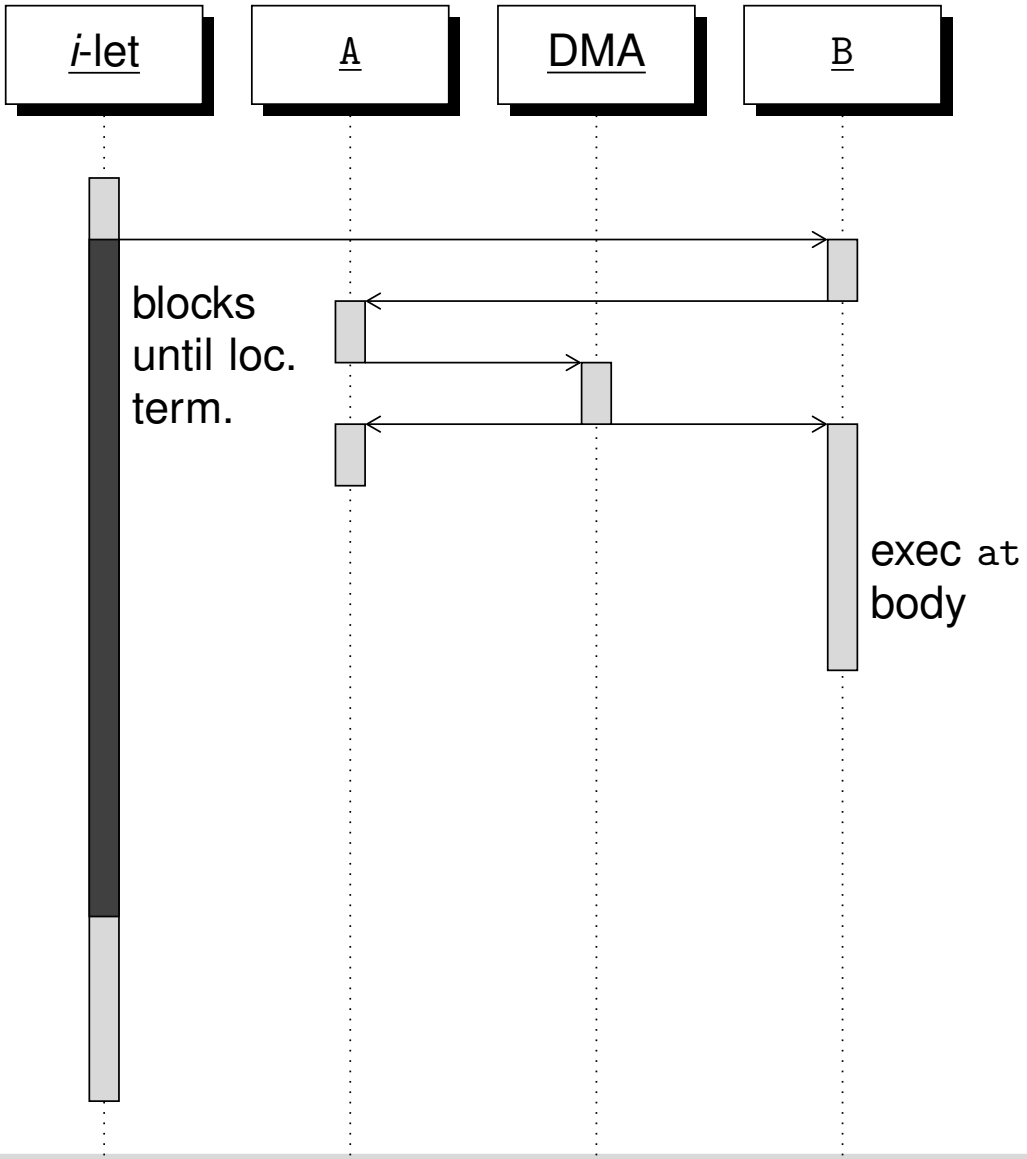
At Expression: `at (B) E`



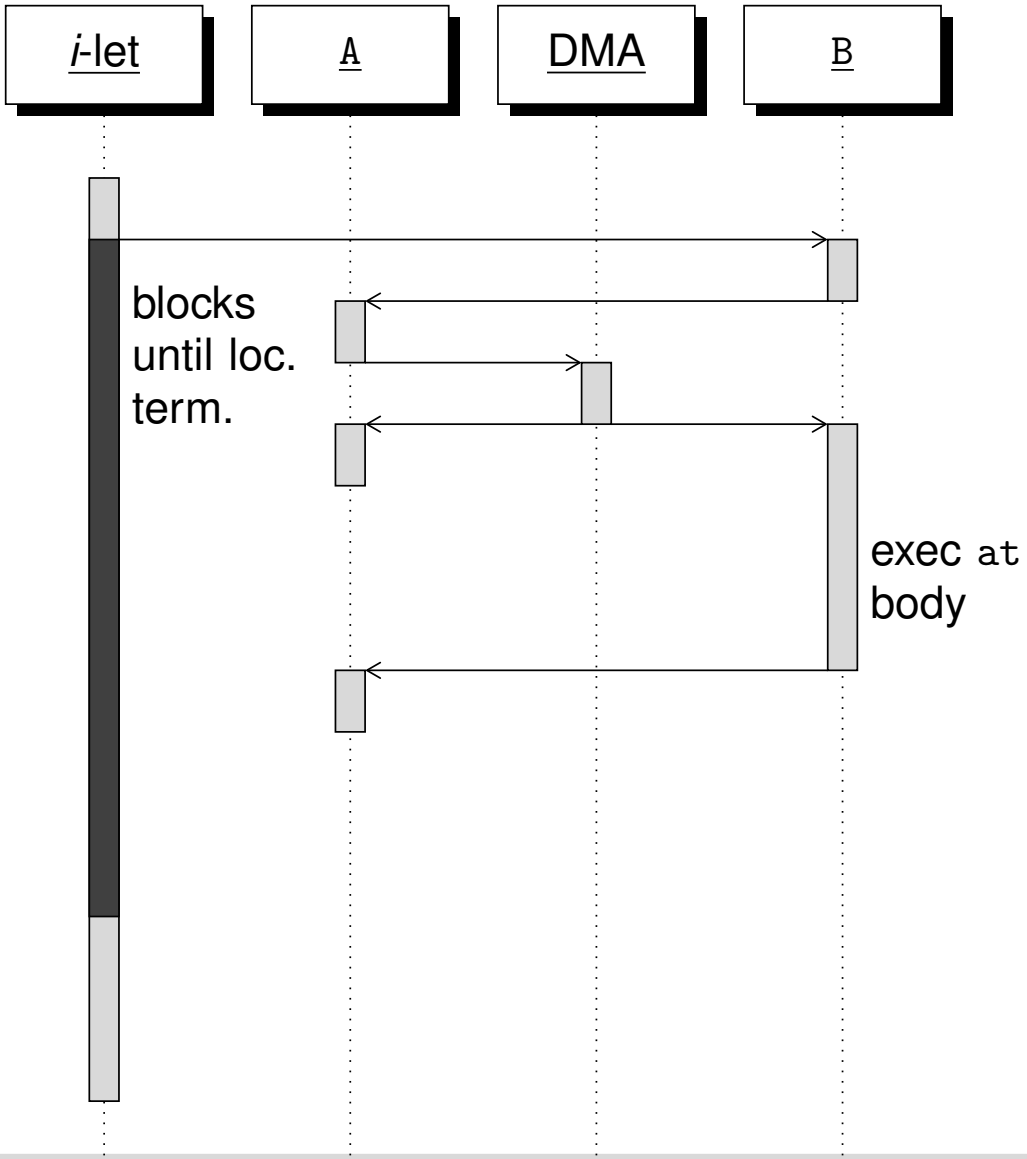
At Expression: `at (B) E`



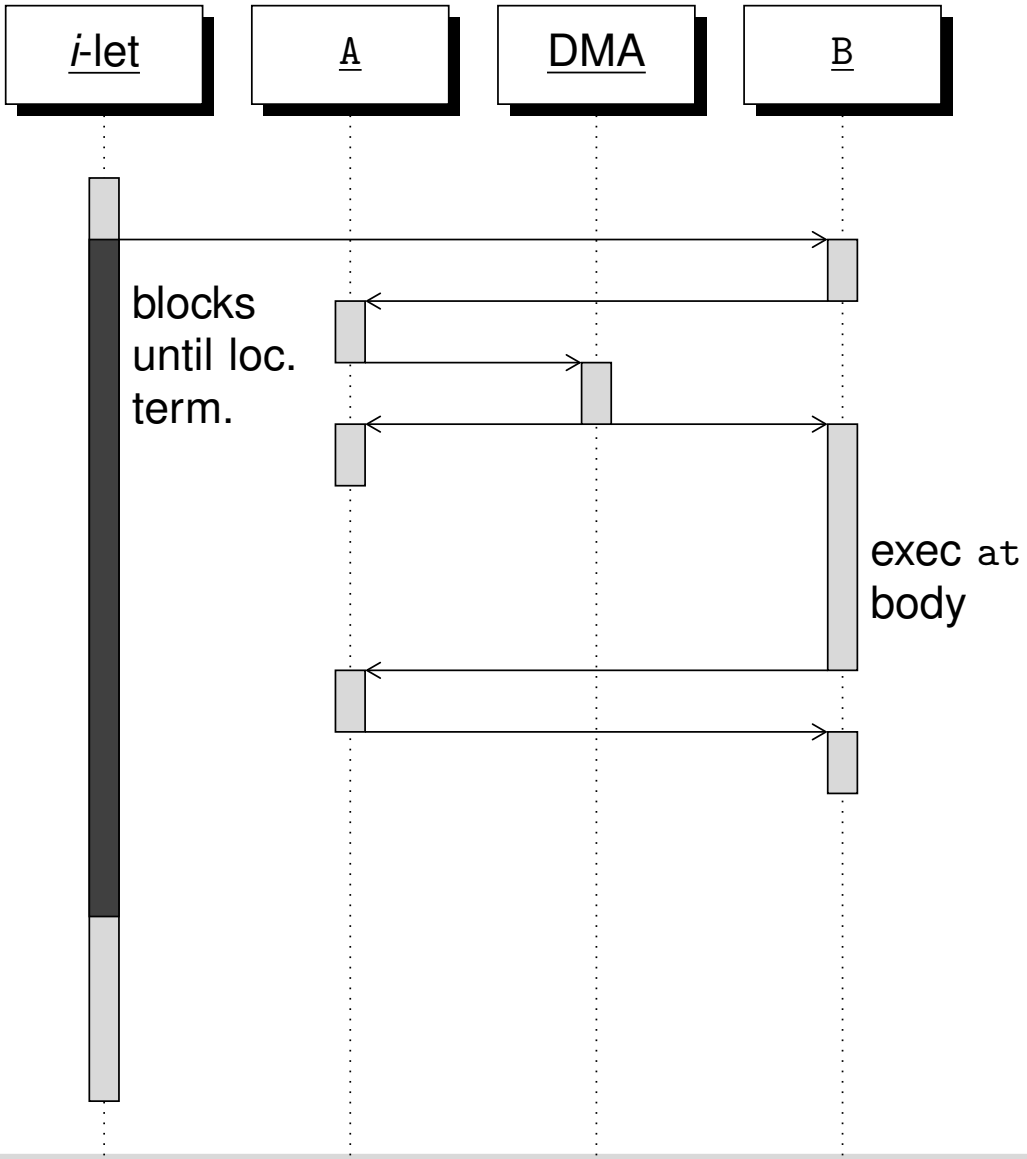
At Expression: `at (B) E`



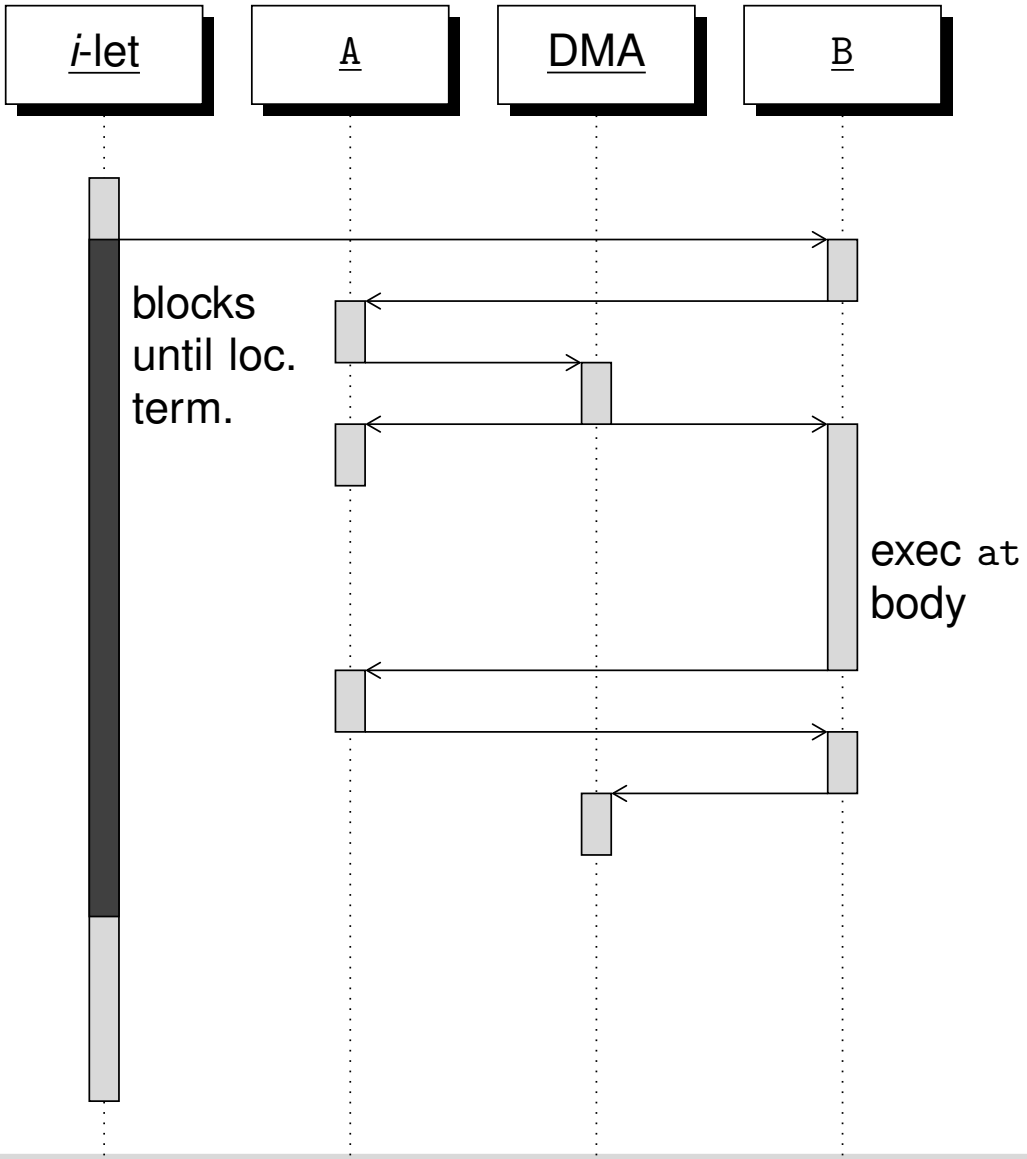
At Expression: `at (B) E`



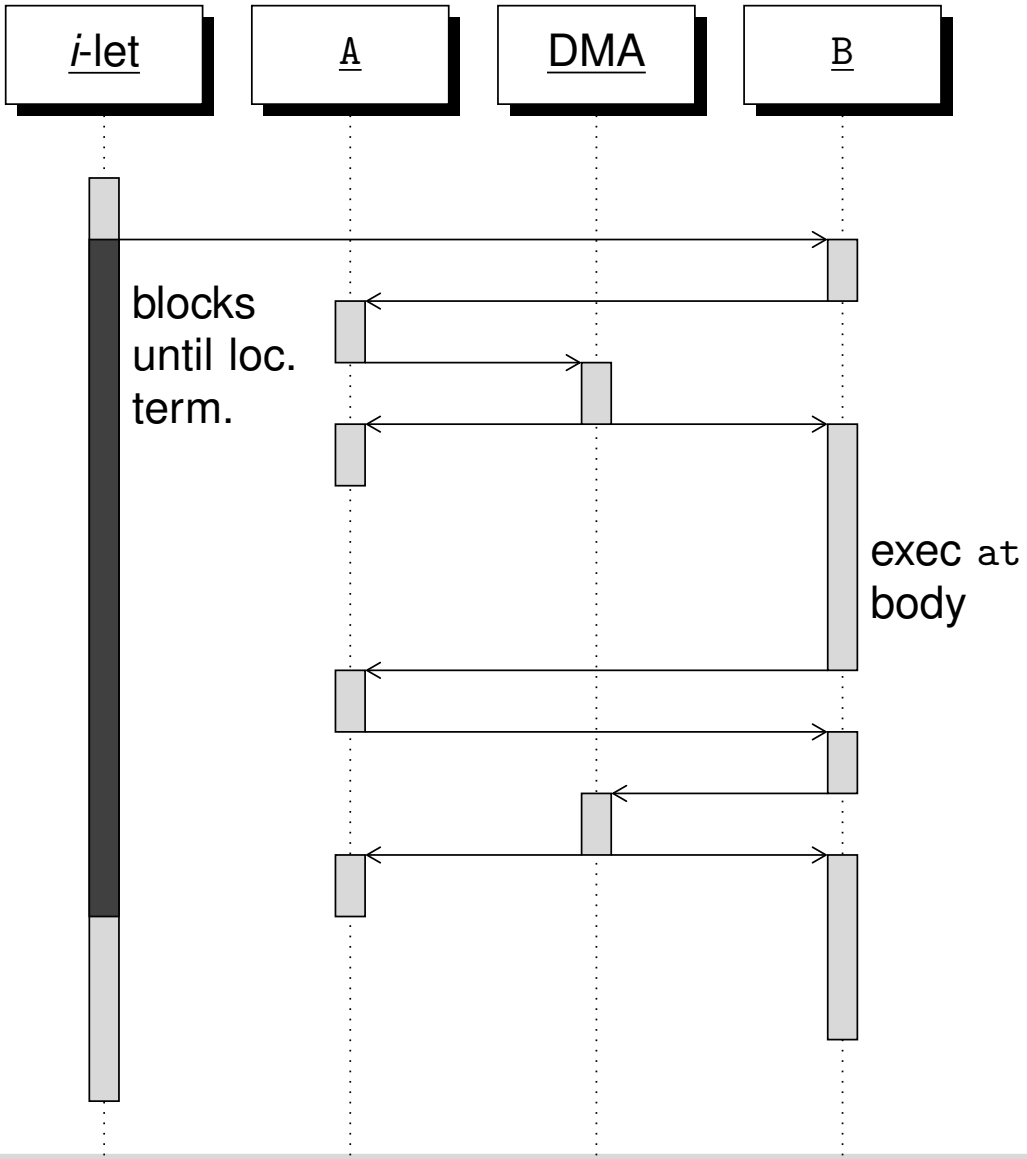
At Expression: `at (B) E`



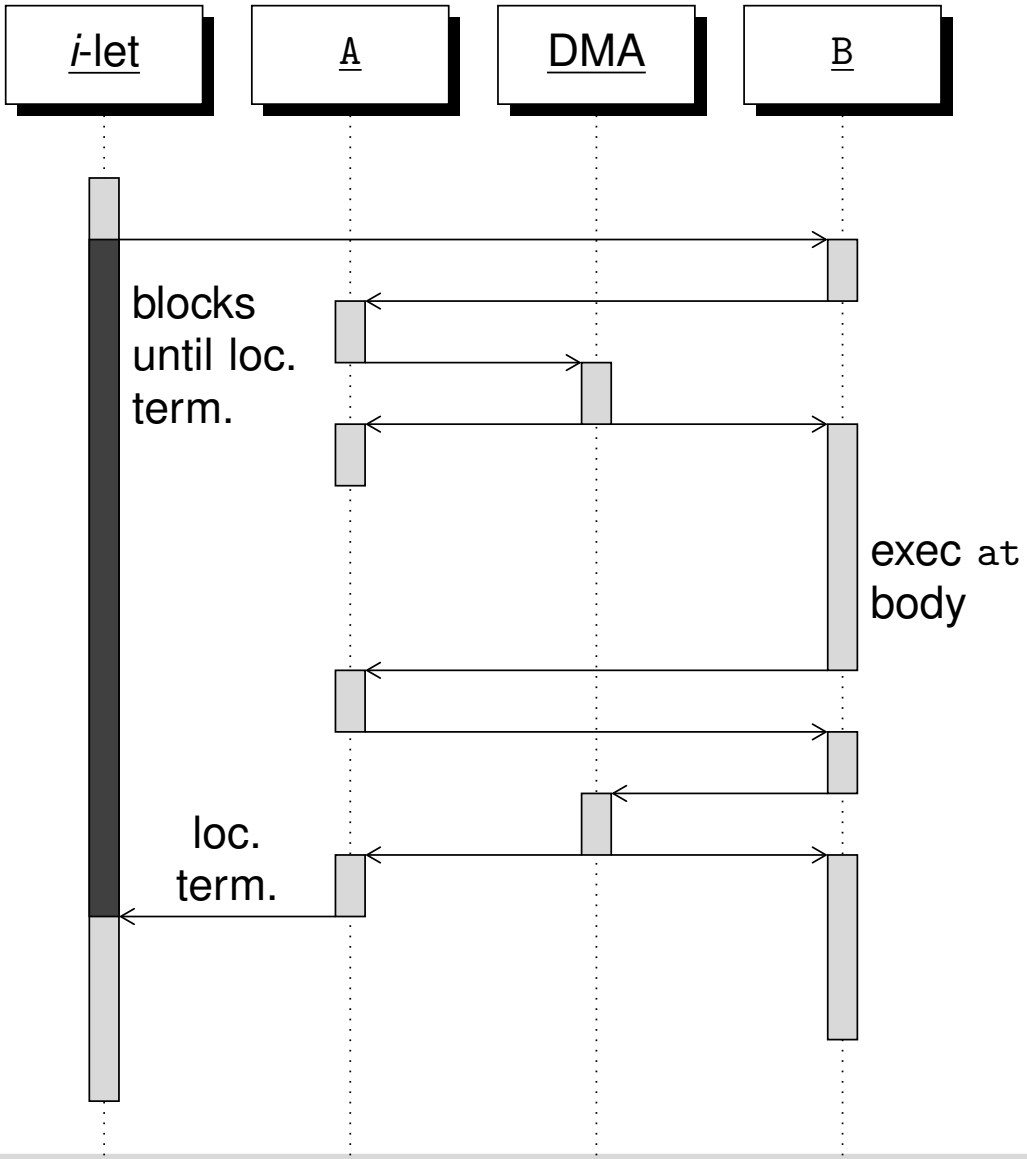
At Expression: `at (B) E`



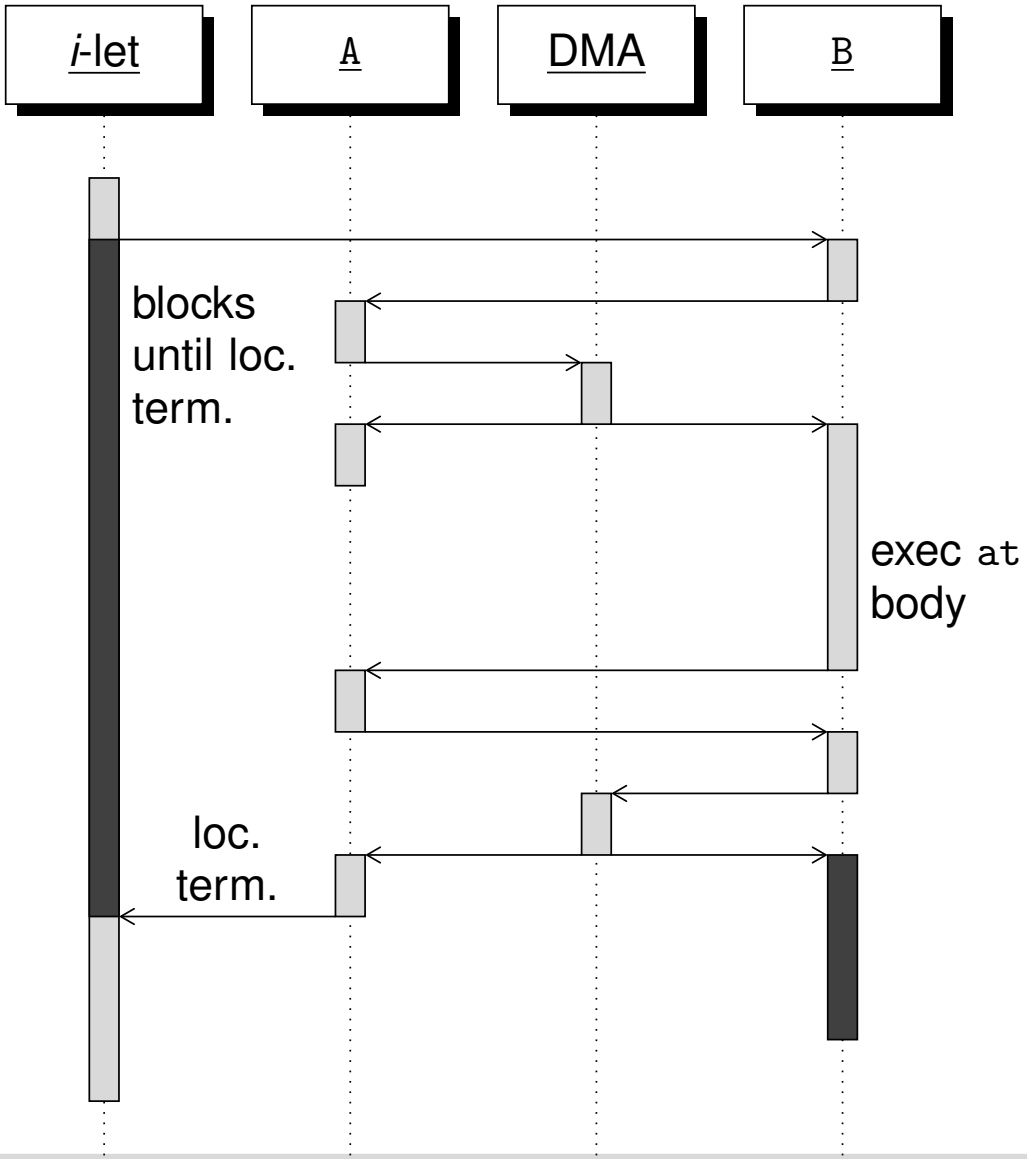
At Expression: `at (B) E`



At Expression: `at (B) E`



At Expression: `at (B) E`



At Expression: `at (B) E`