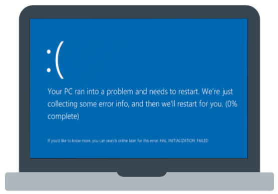
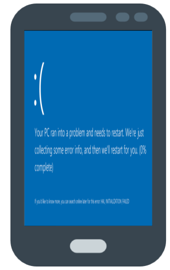
CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim,
Vilhelm Sjöberg, David Costanzo

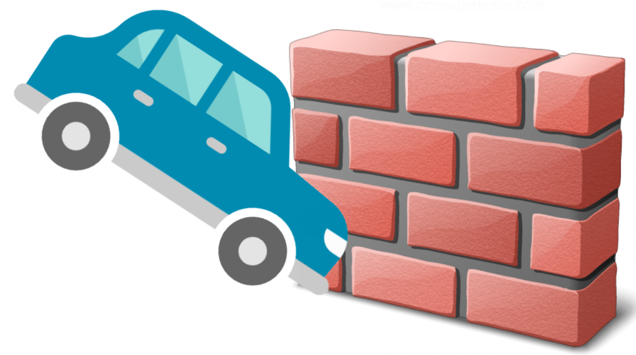
Yale University

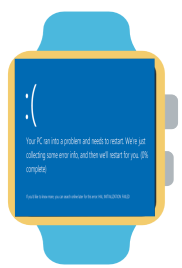

OS Kernel





OS Kernel error





“Complete formal verification is the only known way to guarantee that a system is free of programming errors.”

— seL4 [SOSP’09]

seL4
[SOSP’09]

Verve
[PLDI’10]

Ironclad
[OSDI’14]

mCertiKOS
[POPL’15]

FSCQ
[SOSP’15]

CoGENT
[ASPLOS’16]

...

seL4

[SOSP'09]

Verve

[PLDI'10]

mCertiKOS

[POPL'15]

Ironclad

[OSDI'14]

FSCQ

[SOSP'15]

CoGENT

[ASPLOS'16]

...

verified sequential kernels

seL4
[SOSP'09]

Verve
[PLDI'10]

Ironclad
[OSDI'14]

mCertiKOS
[POPL'15]

FSCQ
[SOSP'15]

CoGENT
[ASPLOS'16]

...

verified software stacks

seL4
[SOSP'09]

Verve
[PLDI'10]

Ironclad
[OSDI'14]

mCertiKOS
[POPL'15]

FSCQ
[SOSP'15]

CoGENT
[ASPLOS'16]

. . .

verified sequential file systems

shared-memory concurrency?

You shall not pass!

shared-memory concurrency?

seL4
[SOSP'09]

Verve
[PLDI'10]

Ironclad
[OSDI'14]

mCertiKOS
[POPL'15]

FSCQ
[SOSP'15]

CoGENT
[ASPLOS'16]

. . .

seL4

[SOSP'09]

"Proofs about concurrent programs are hard, much harder than proofs about sequential programs."

hard

seL4
[SOSP'09]

Verve
[PLDI'10]

Ironclad
[OSDI'14]

mCertiKOS
[POPL'15]

CoGENT
[ASPLOS'16]

FSCQ
[SOSP'15]

. . .

FSCQ

[SOSP'15]

hard

"[…]multiprocessor support, which may require global changes […]"

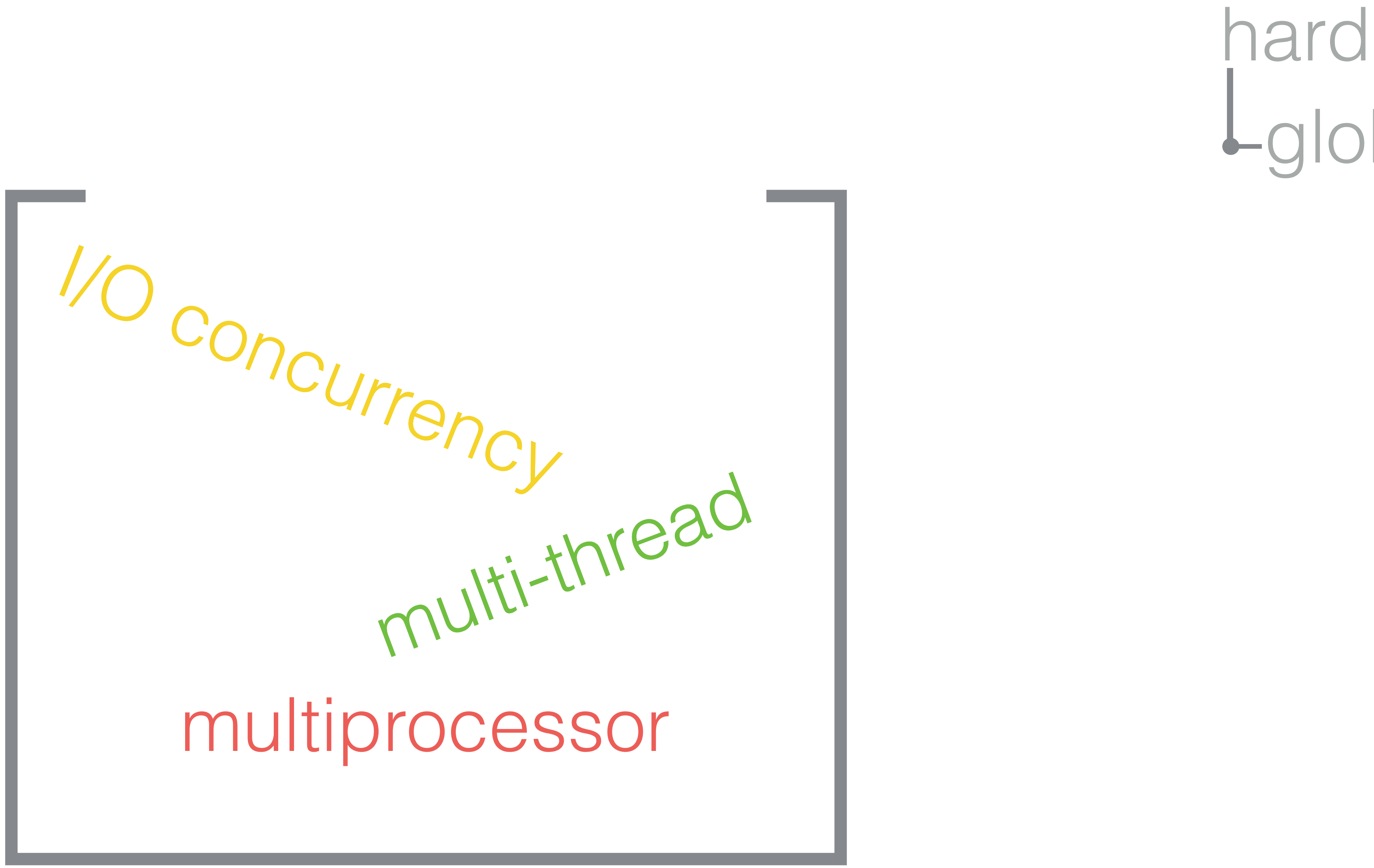
FSCQ

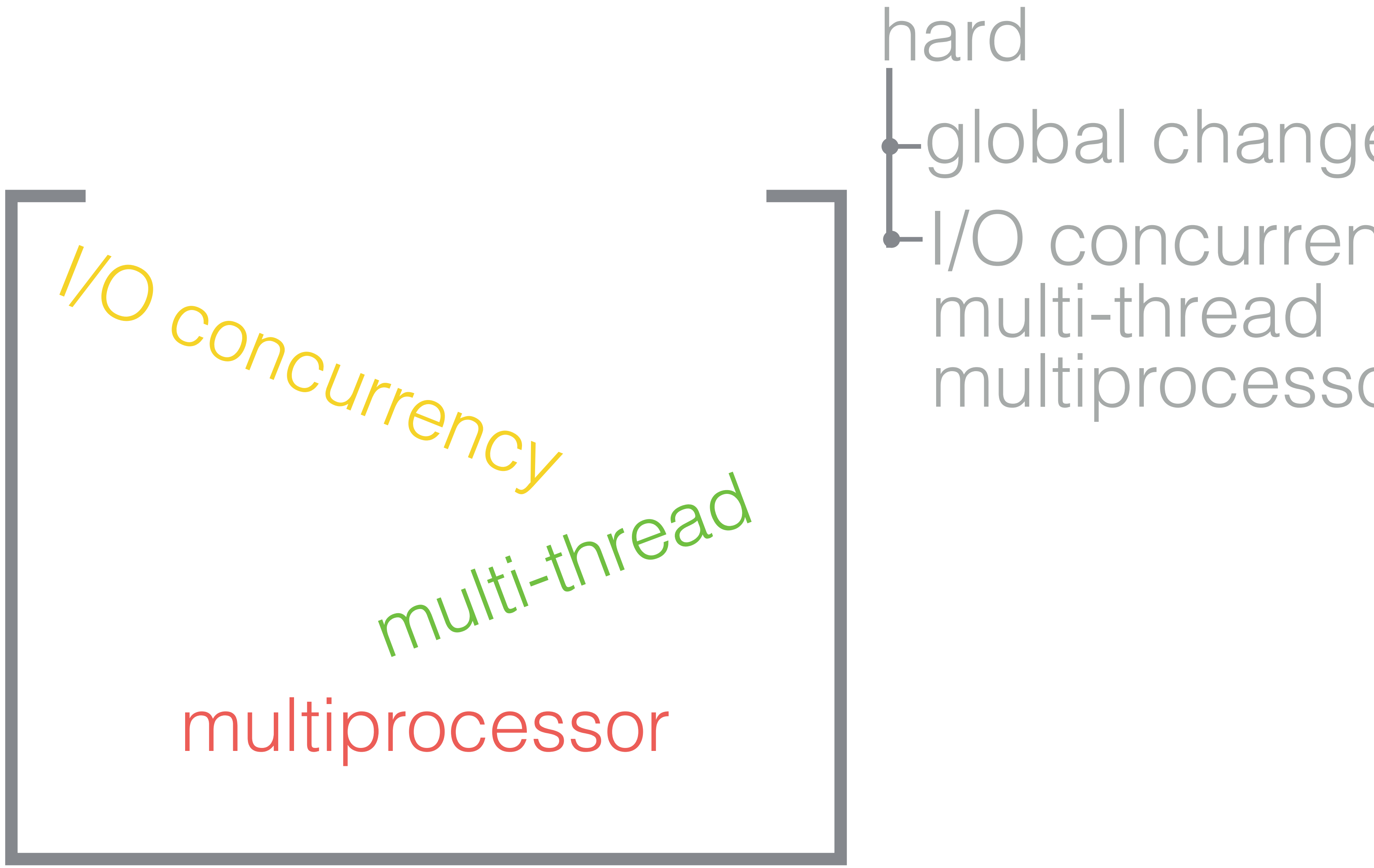
[SOSP'15]

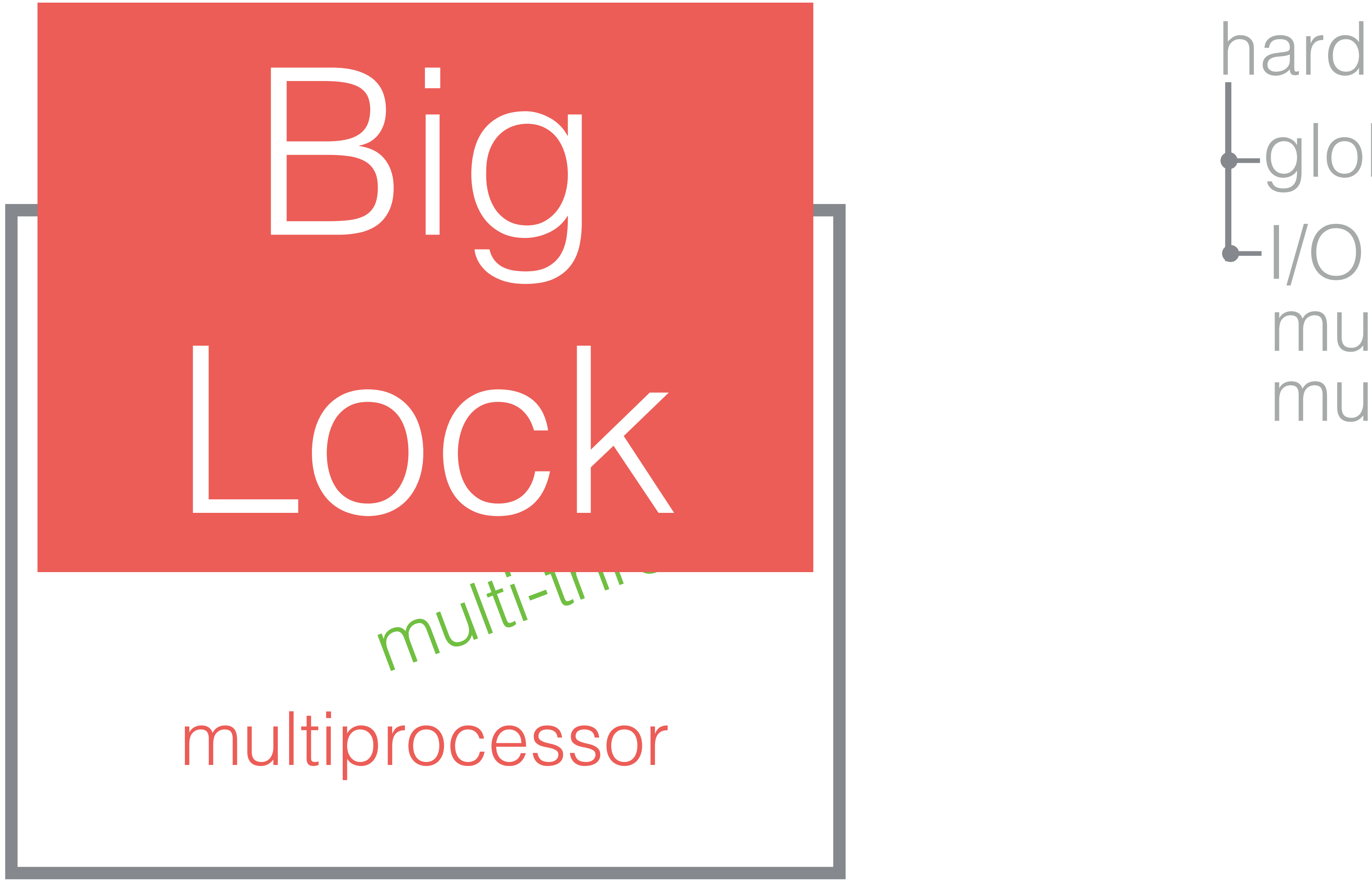
hard

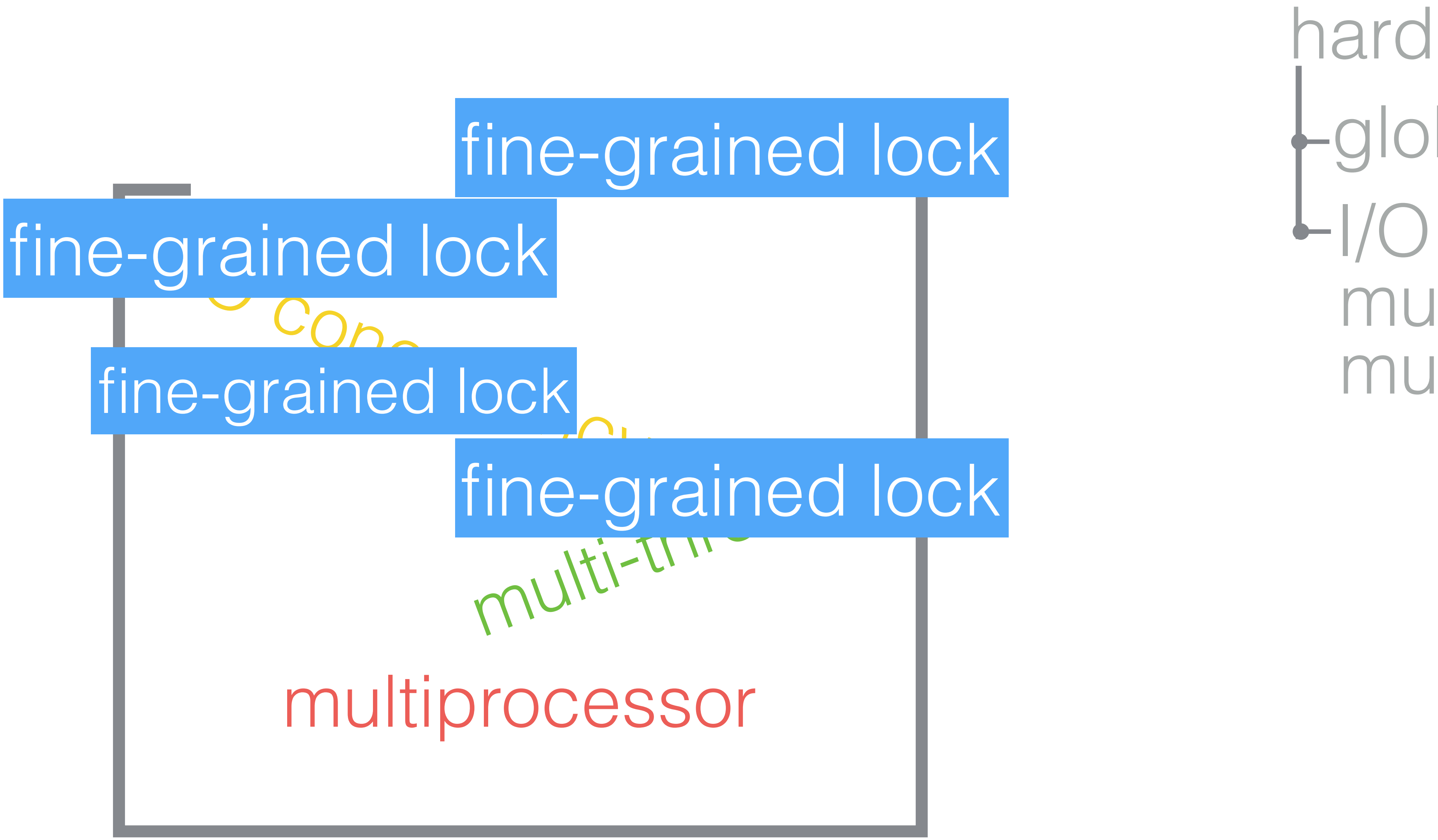
global changes

"[…]multiprocessor support, which may require global changes […]"


hard
glob
I/O concurrency
multi-thread
multiprocessor


I/O concurrency
multi-thread
multiprocessor
hard
global change
I/O concurren
multi-thread
multiprocesso


Big Lock
multiprocessor
hard
I/O


fine-grained lock
fine-grained lock
fine-grained lock
fine-grained lock
multiprocessor
hard
I/O

S.Peters et al.

[APSys'15]

hard

glol

I/O

mu

mu

“

the verification to a kernel version with fine-grained locking will far exceed the cost already paid for verifying the single core version.

”

S.Peters et al.

[APSys'15]

"the verification to a kernel version with fine-grained locking will far exceed the cost already paid for verifying the single core version."

hard

- global cha
- I/O concu multi-threa multiproce
- fine-grain

What to prove?

functional correctness

liveness system calls will eventually return

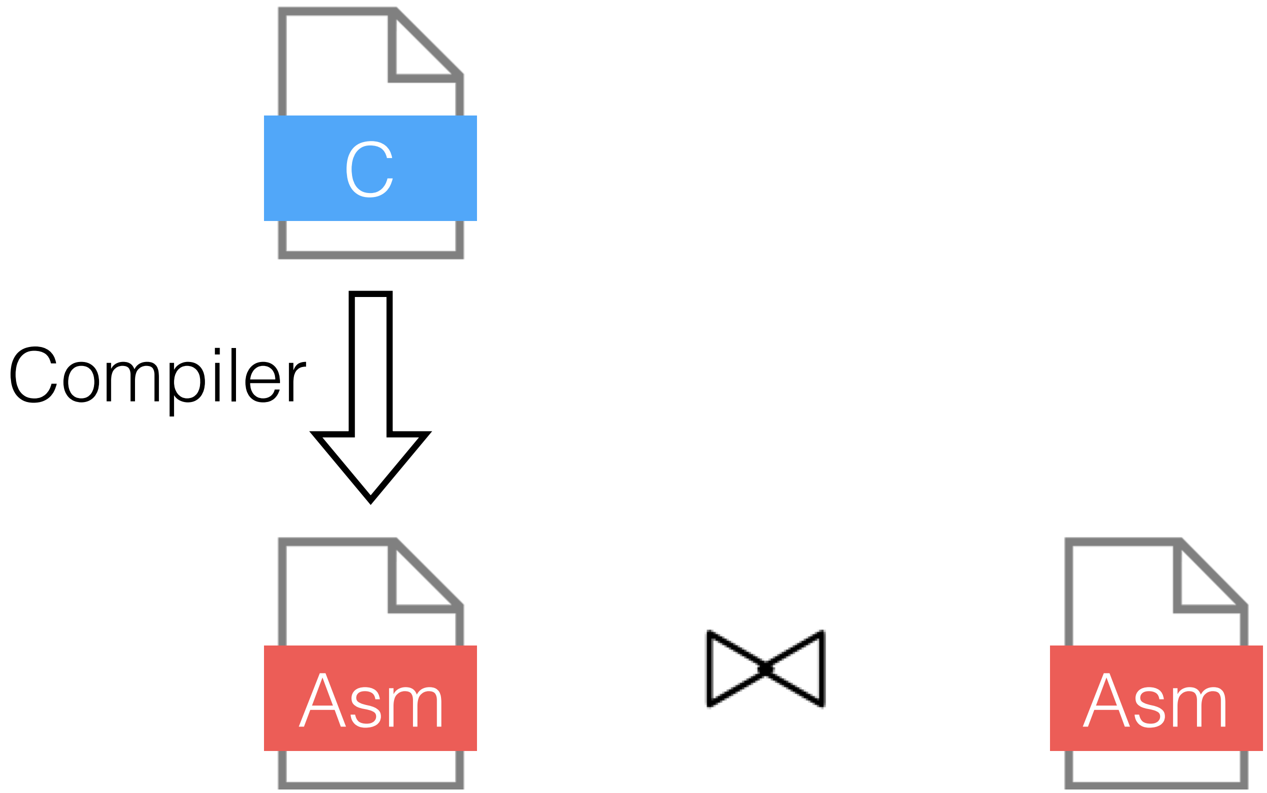
hard
glob
I/O
mul
mul
fine

concurrent OS kernel

hard

- global change
- I/O concurren
  multi-thread
  multiprocesso
- fine-grained lo
- liveness

concurrent OS kernel



hard
glob
I/O
mul
mu
fine
live

hard

- global changes
- I/O concurrency

cost

- asm&C
- compiler

hard

- global changes
- I/O concurrency
  multi-thread
  multiprocessor
- fine-grained lock
- liveness
- asm&C
- compiler
- cost

CertiKOS

solves all these challenges

hard

glob

I/O

mul

mu

fine

live

asn

cor

cos

contributions

CertiKOS

hard

glob

I/O

mul

mu

fine

live

asr

cor

cos

mC2, the first formally verified concurrent OS kernel with fine-grained locks.

contributions

- mC2
- fine-grained lock

CertiKOS

hard

- glob
- I/O
  mul
  mu
- live
- asn
- con
- cos

mC2, the first formally verified concurrent OS kernel with fine-grained locks.

CertiKOS

contributions

- mC2
- fine-grained lock

hard

- glob
- I/O
  mul
  mu
- live
- asr
- cor
- cos

both functional correctness
and liveness

CertiKOS

contributions
- mC2
- fine-grained lock
- liveness

hard
- glob
- I/O
mul
mu
- asr
- cor
- cos

both functional correctness
and liveness

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness

certified concurrent layers

hard

- glob
- I/O
  mul
  mu
- asn
- cor
- cos

CertiKOS

contributions
- mC2
- fine-grained lock
- liveness

hard
- glob
- I/O
  mul
  mu
- asn
- cor
- cos

reuses sequential verification techniques.

certified concurrent layers

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- global changes

hard

- I/O
- mul
- mu
- asn
- cor
- cos

reuses sequential verification techniques.

certified concurrent layers

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs

hard

- I/O
- mul
- mu
- asr
- cor
- cos

handles all 3 kinds of concurrency

certified concurrent layers

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- I/O concurrency
  multi-thread
  multiprocessor

handles all 3 kinds of concurrency

certified concurrent layers

hard

- asr
- cor
- cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3

C 6100 LOC

Asm 400 LOC

hard

- asr
- cor
- cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C

C 6100 LOC

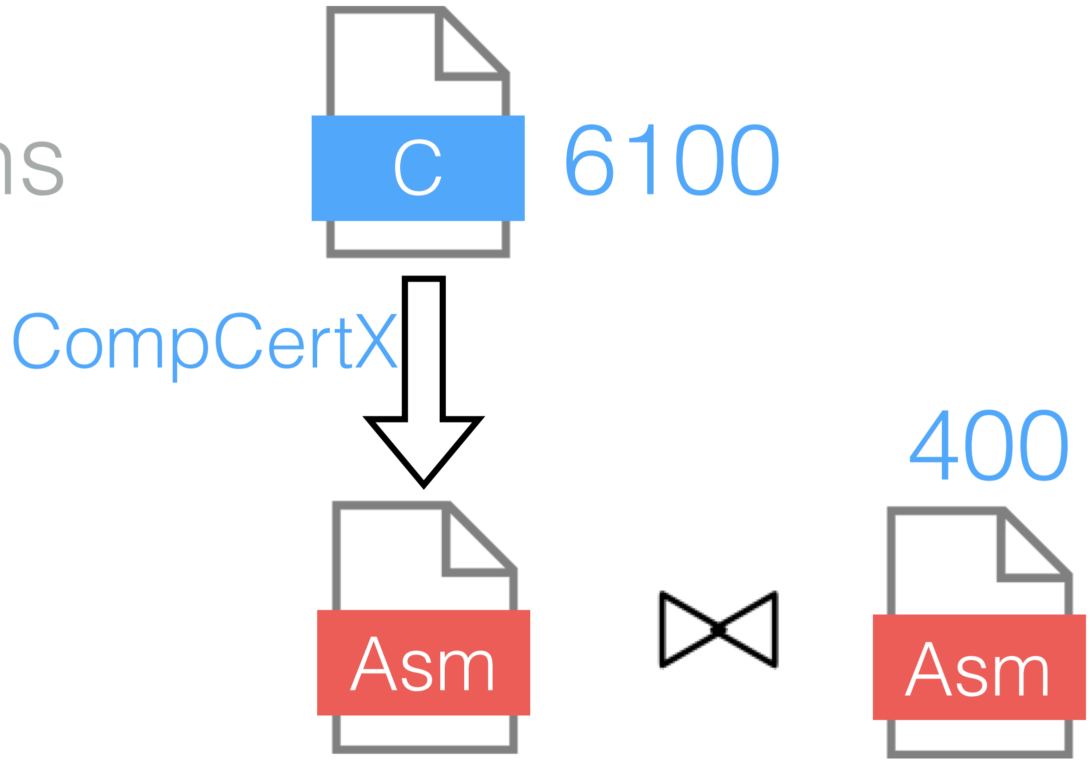
Asm 400 LOC

hard

- cor
- cos

CertiKOS

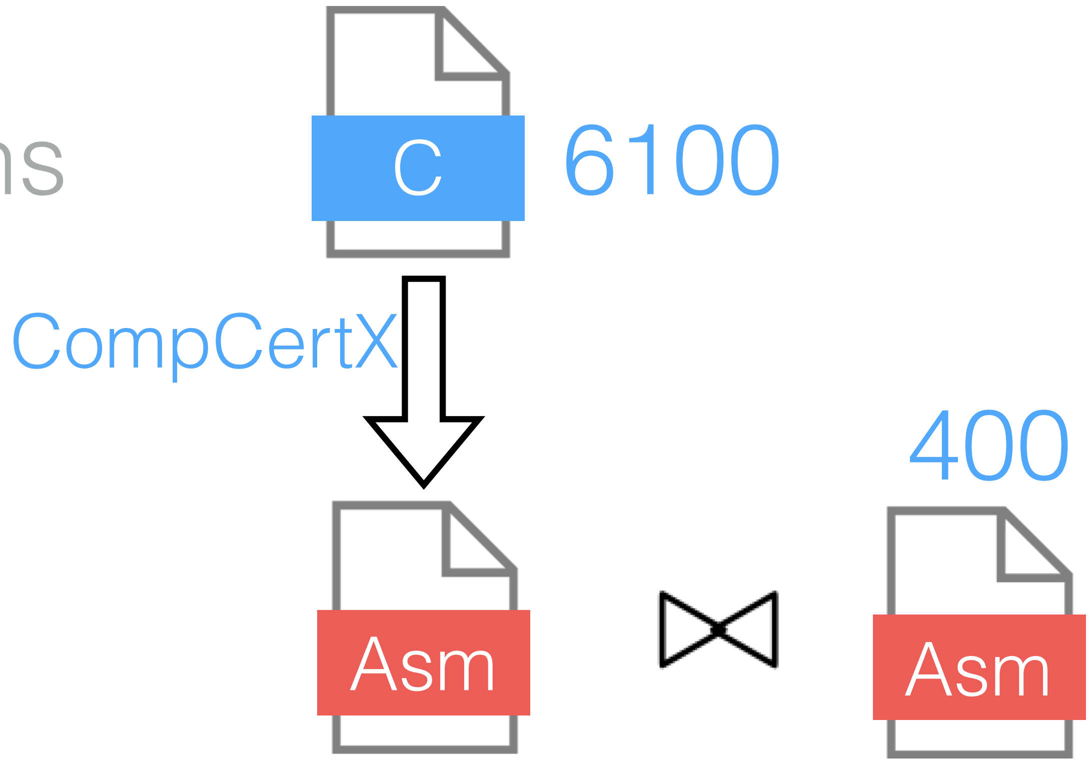
contributions
- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C



hard
- co
- cos

CertiKOS

contributions
- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- compiler



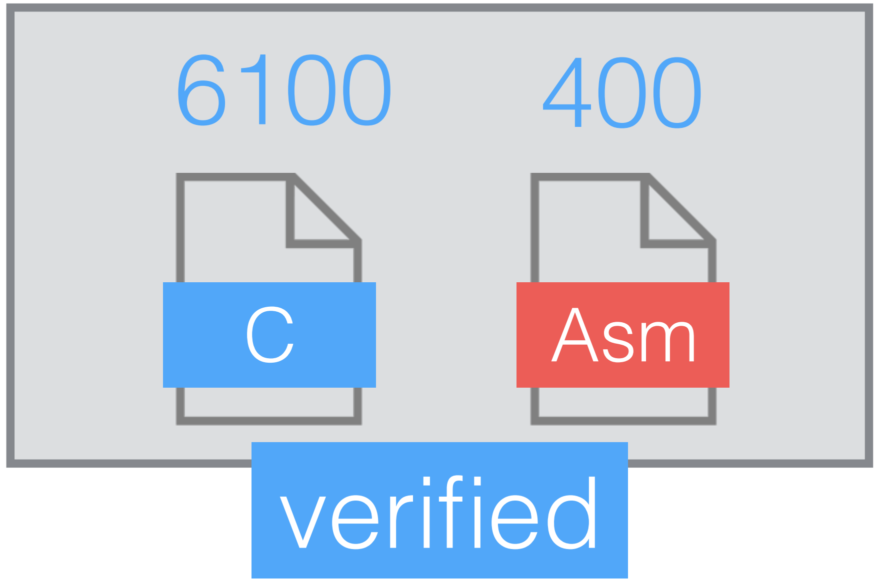
hard

cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- CompCertX



~~model checking~~ Coq ~~SMT solver~~

machine-checkable proof

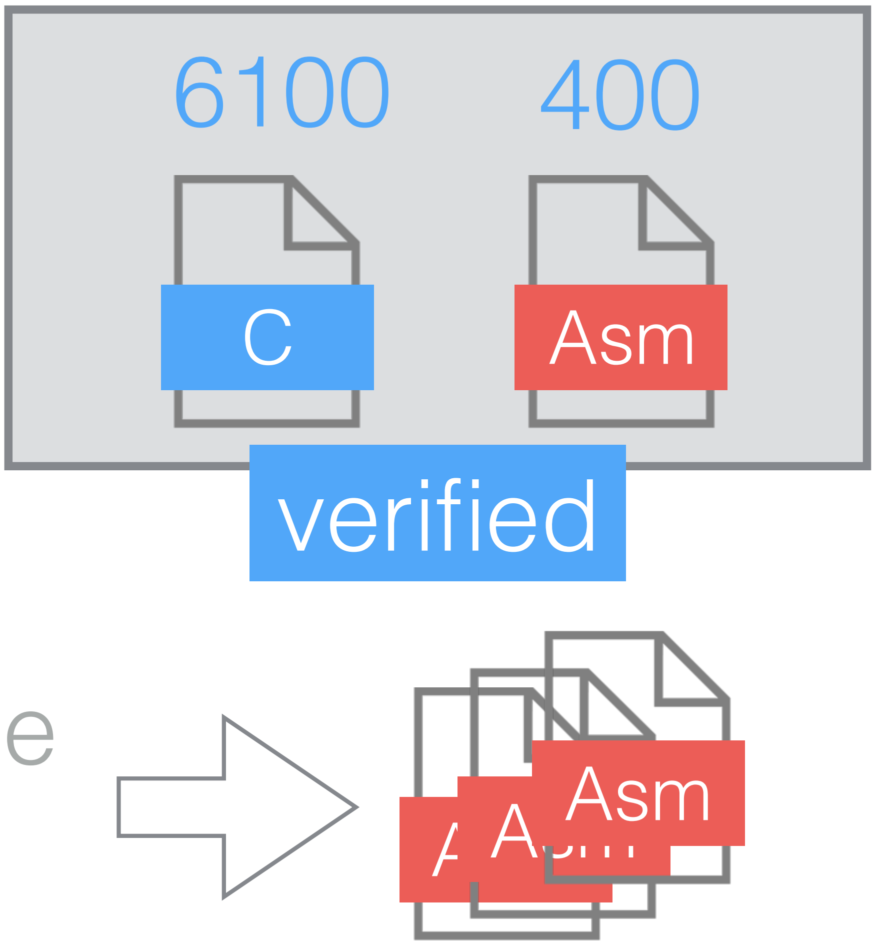
hard

cos

CertiKOS

contributions

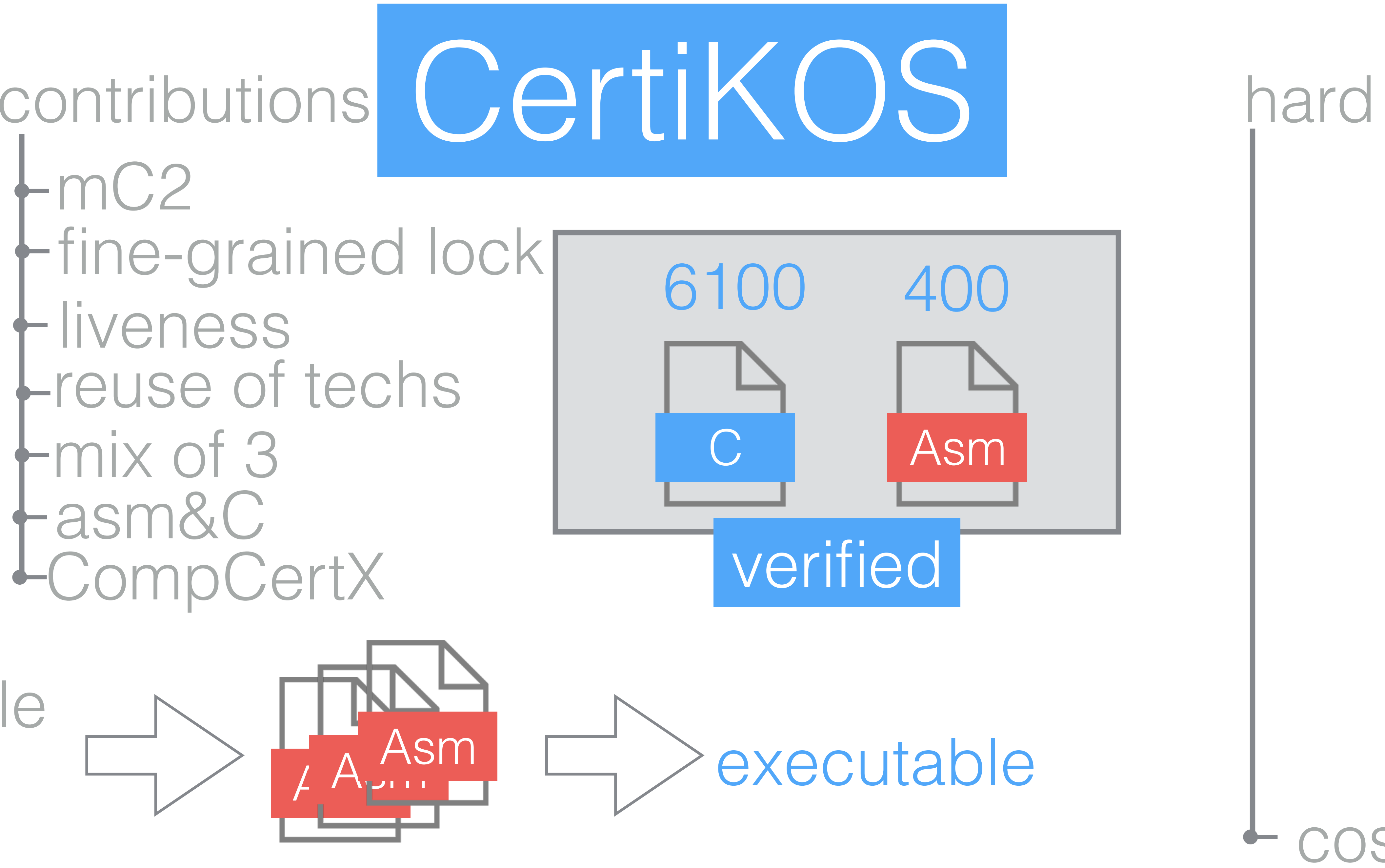
- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- CompCertX



machine-checkable proof

hard

cos

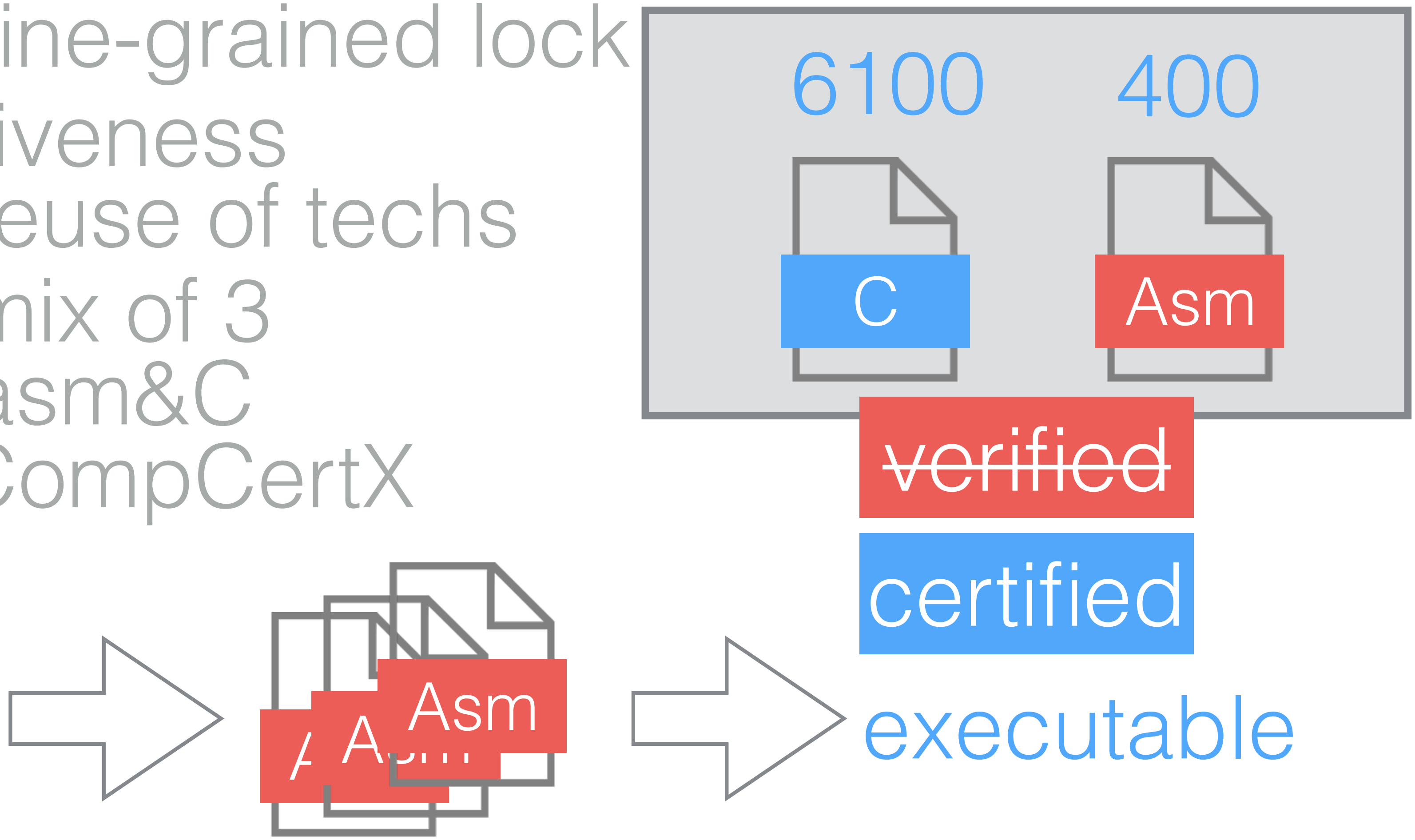

CertiKOS
contributions
mC2
fine-grained lock
liveness
reuse of techs
mix of 3
asm&C
CompCertX
hard
6100
400
C
Asm
verified
le
Asm
executable
cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- CompCertX

hard



cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- CompCertX
- certified

mCertiKOS 1 py

[POPL'15]

+ extensions 0.5 py

+ device 0.5 py

[PLDI'16]

+ concurrency 2 py

hard

cos

CertiKOS

contributions

- mC2
- fine-grained lock
- liveness
- reuse of techs
- mix of 3
- asm&C
- CompCertX
- extensibility
- certified
- cost

new technical contributions

certified concurrent layers

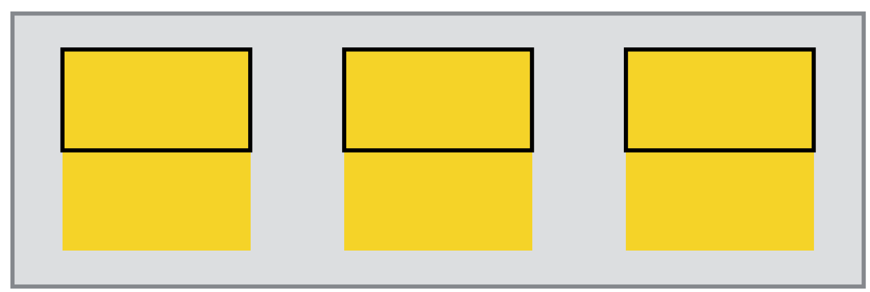
logical log + hardware scheduler
+ environment context

push/pull model

multicore machine lifting

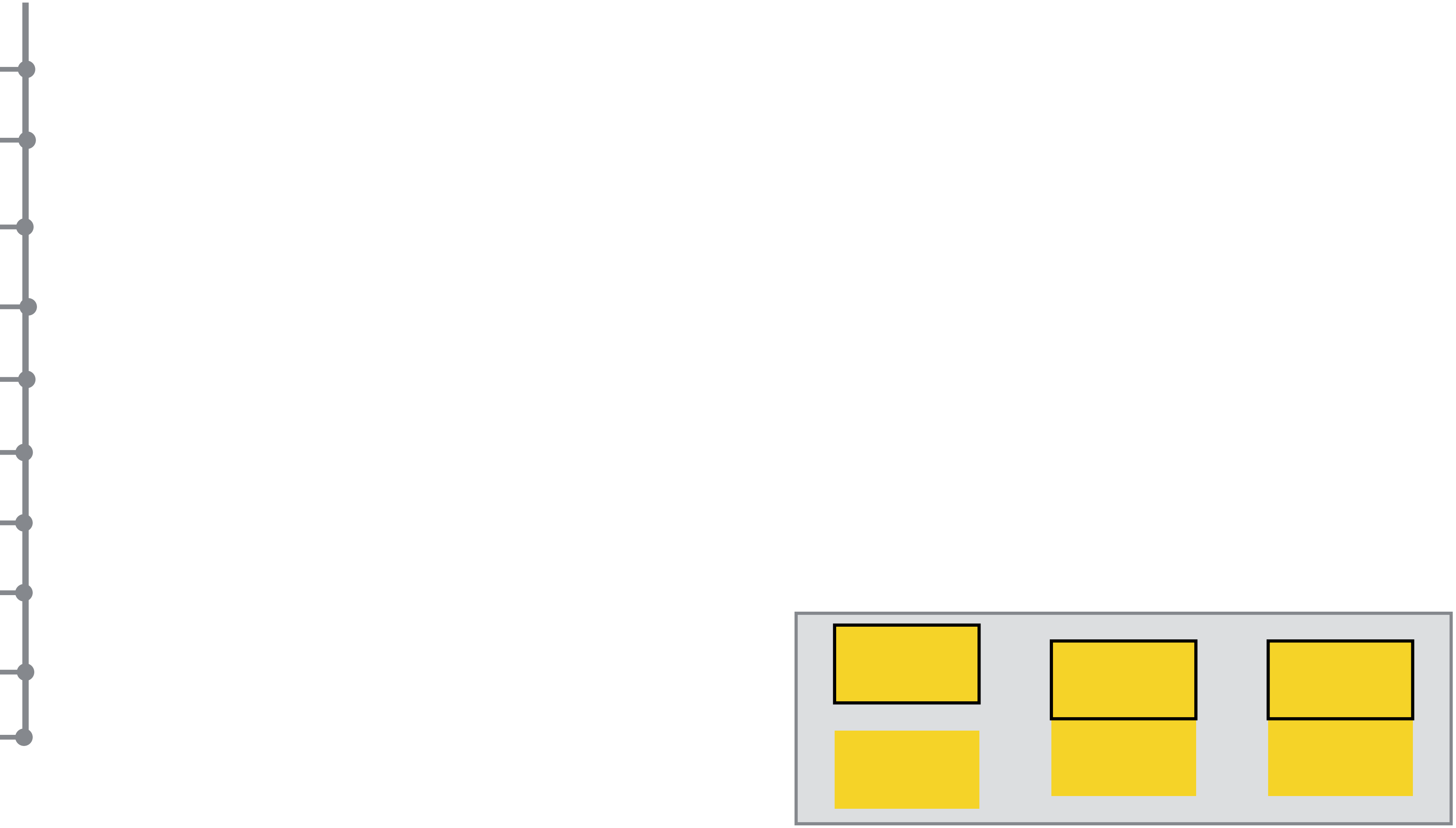
certified sequential layers

certified objects

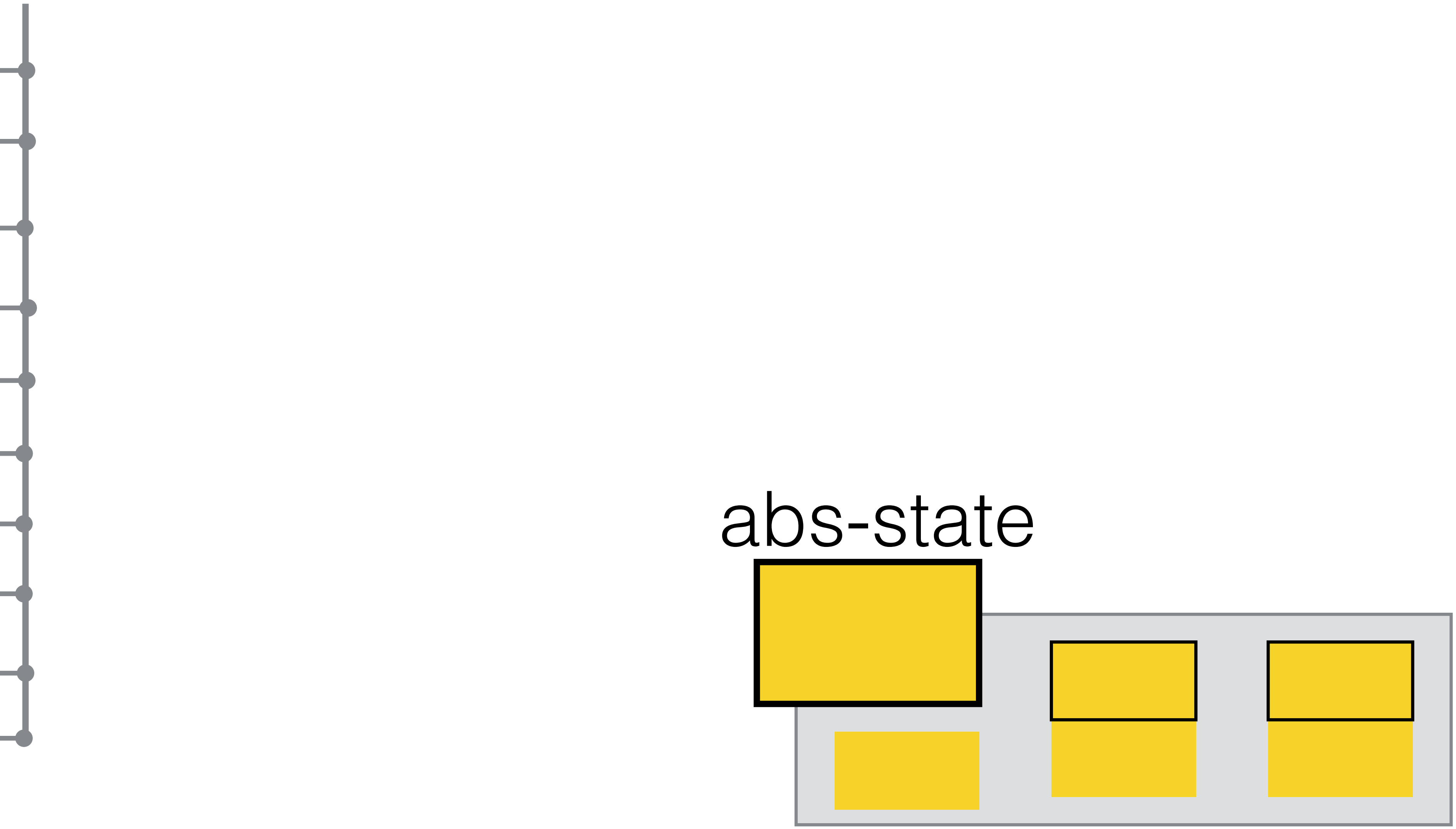


specification of modules to trust

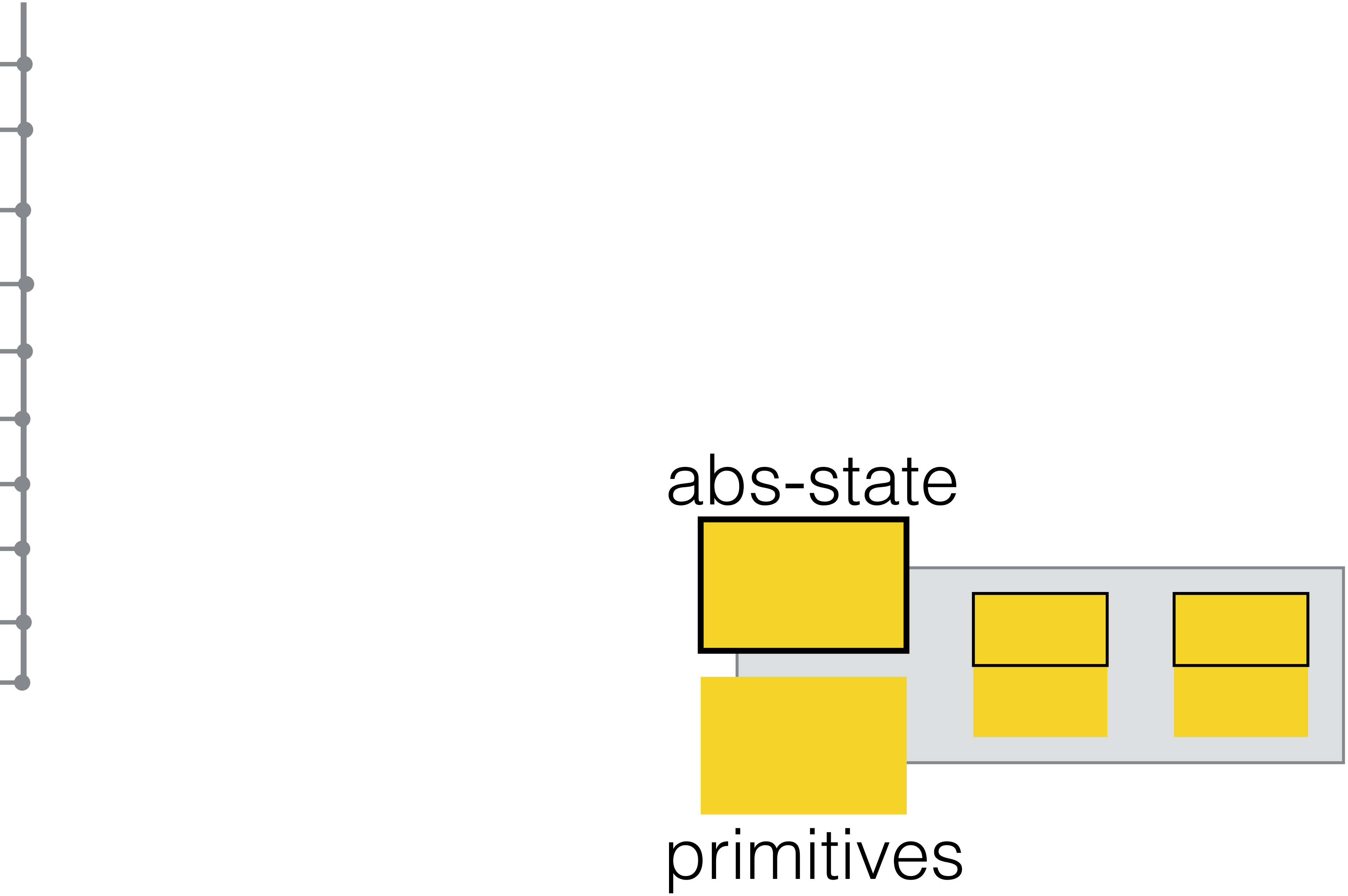
certified sequential layers

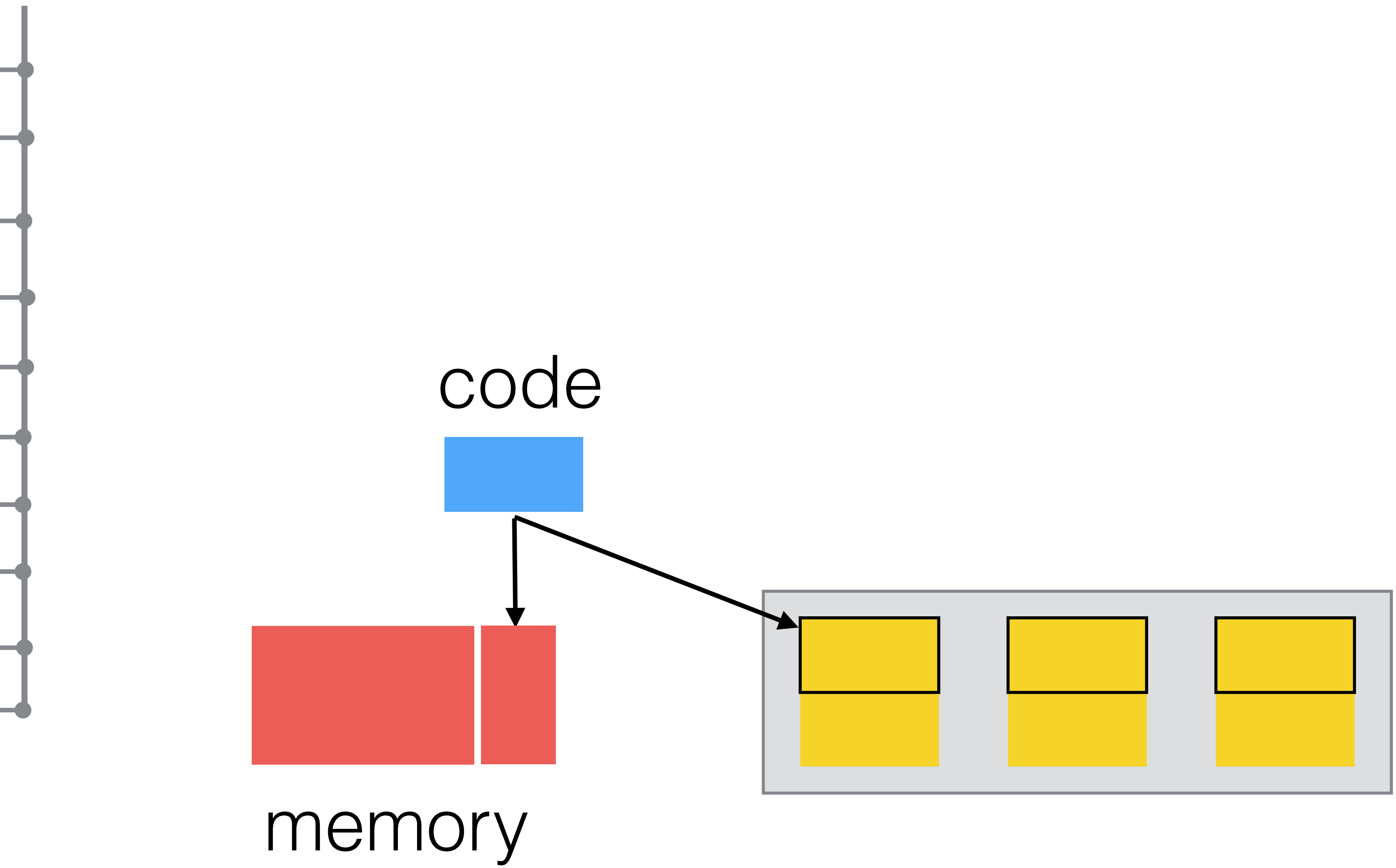


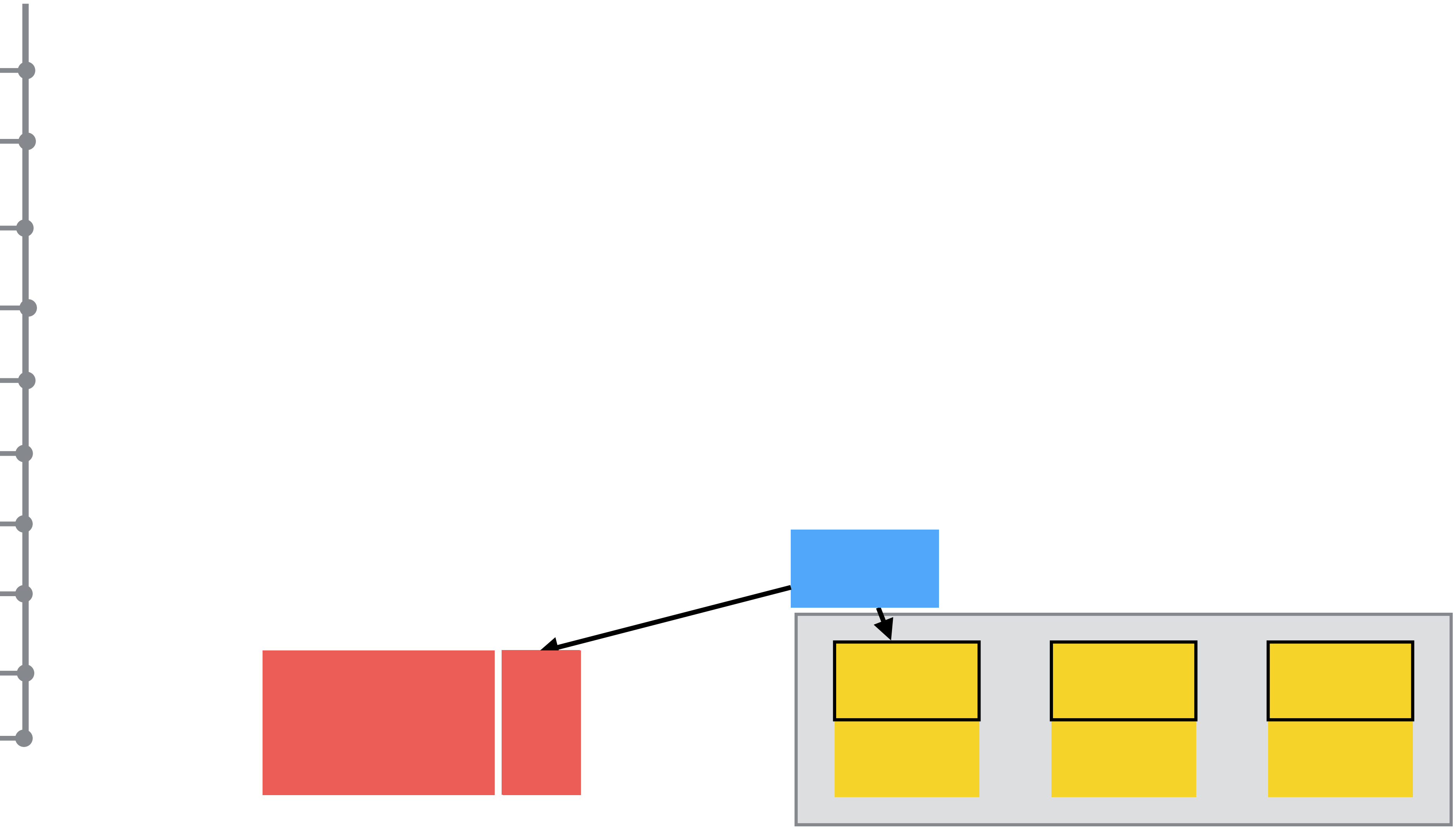
certified sequential layers

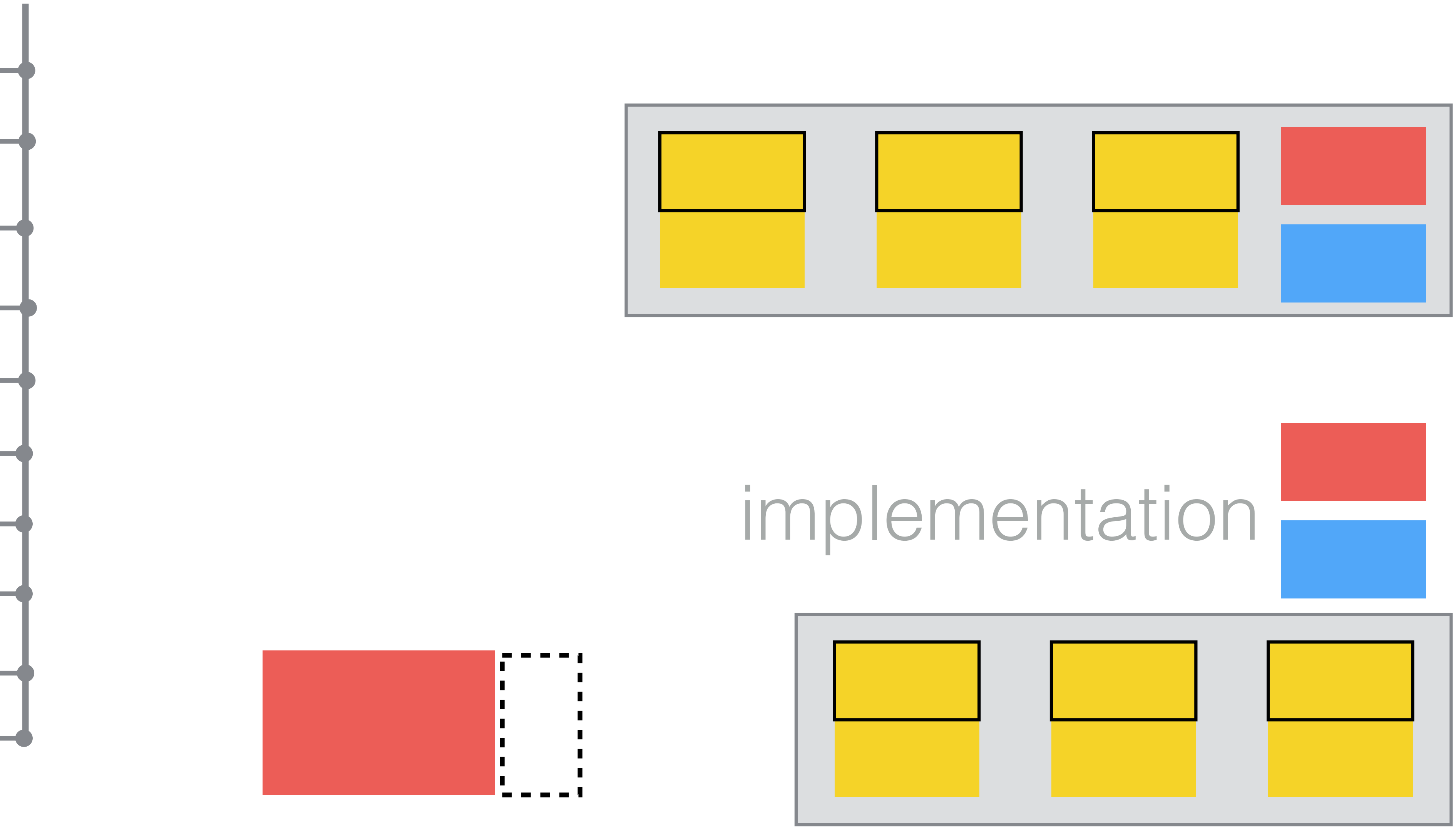


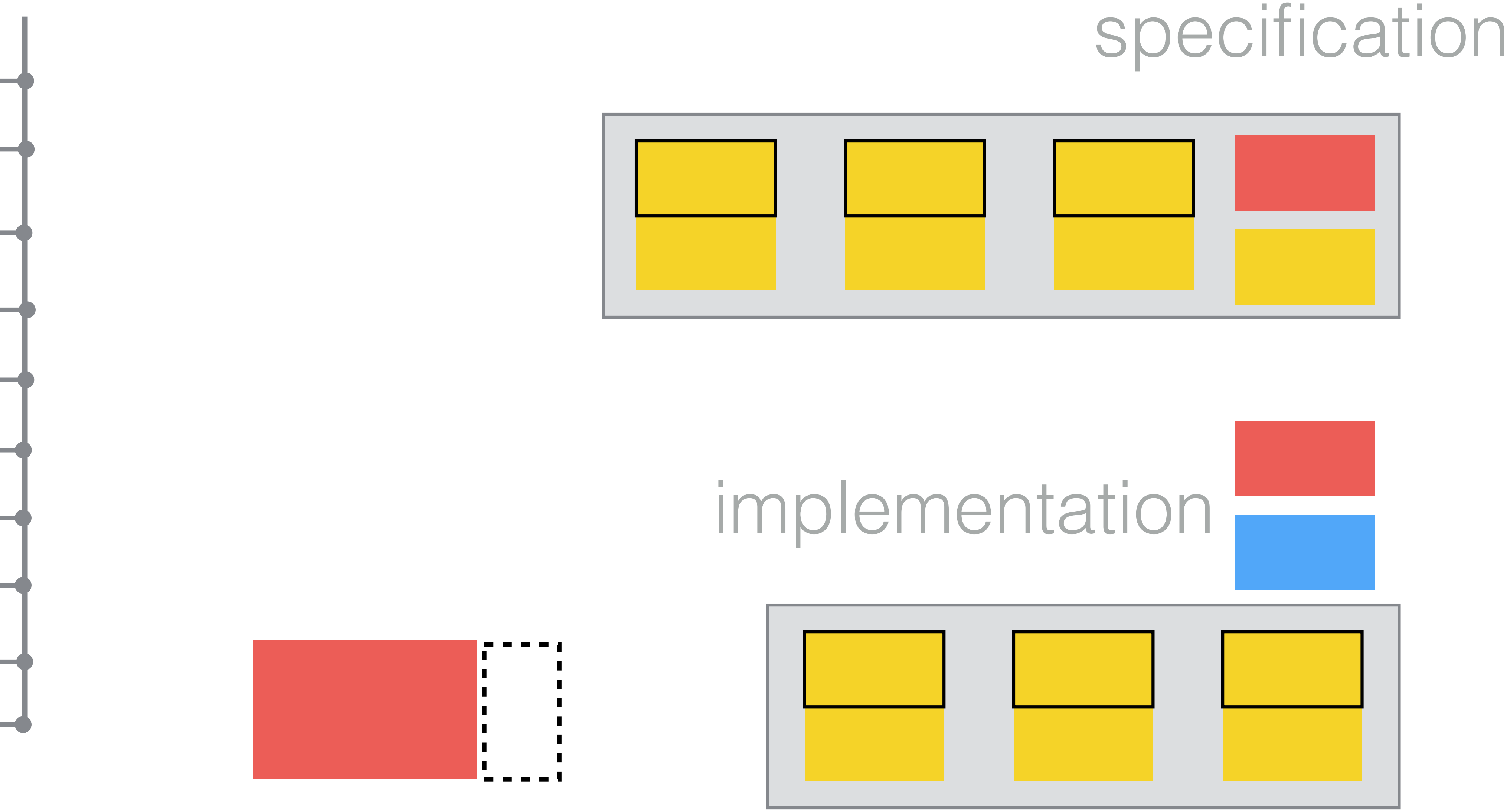
certified sequential layers

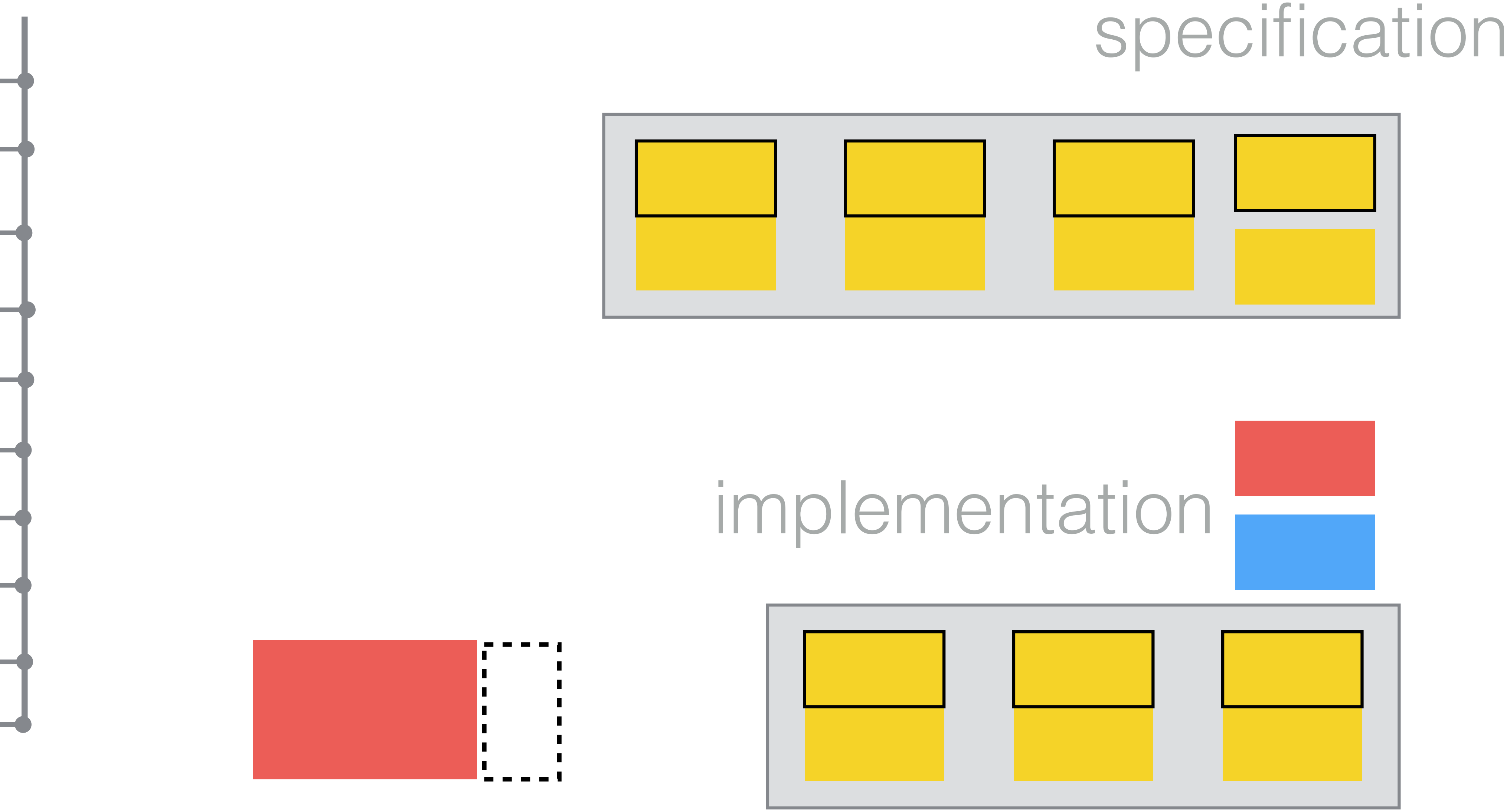



code
memory

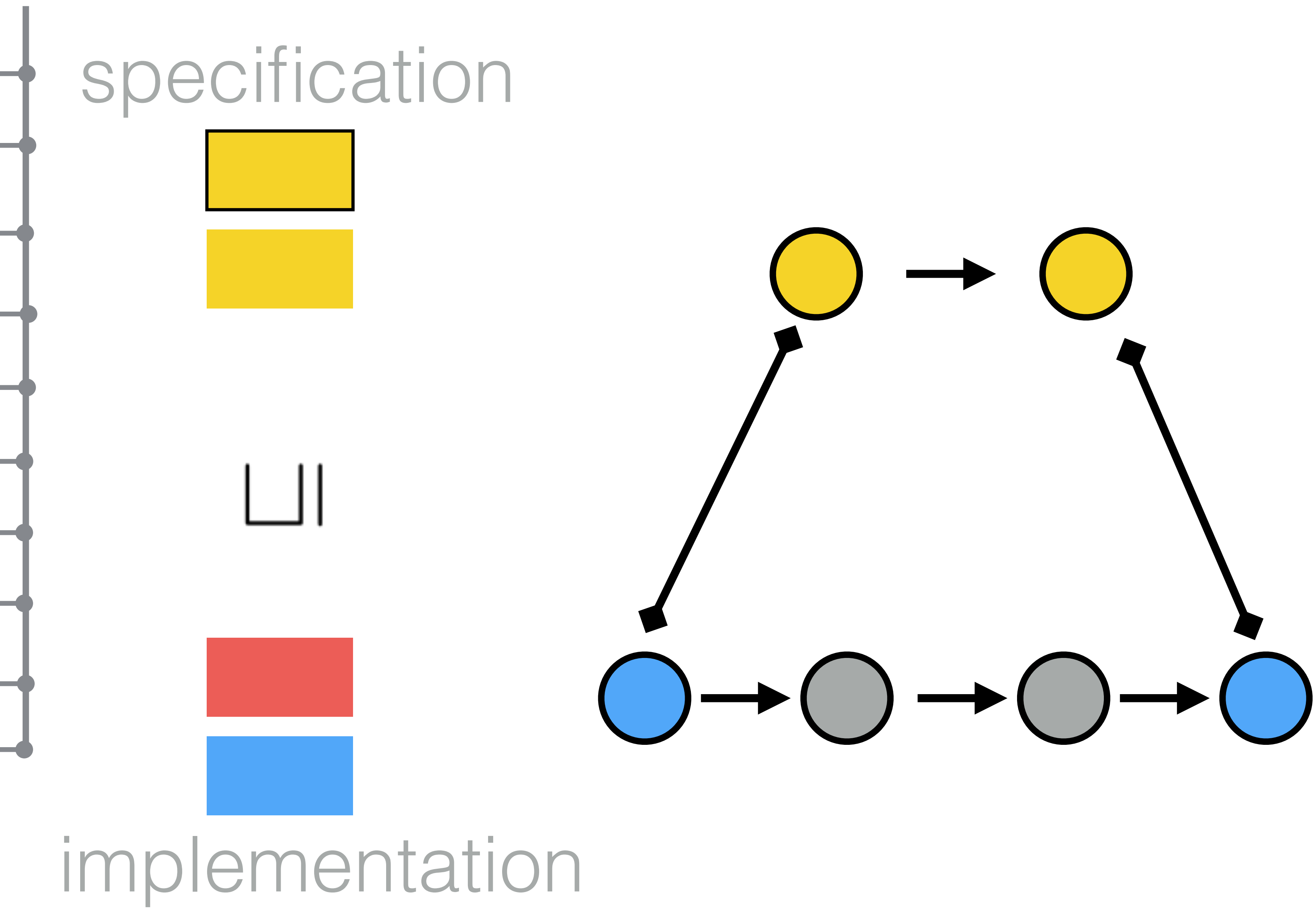



implementation


specification
implementation

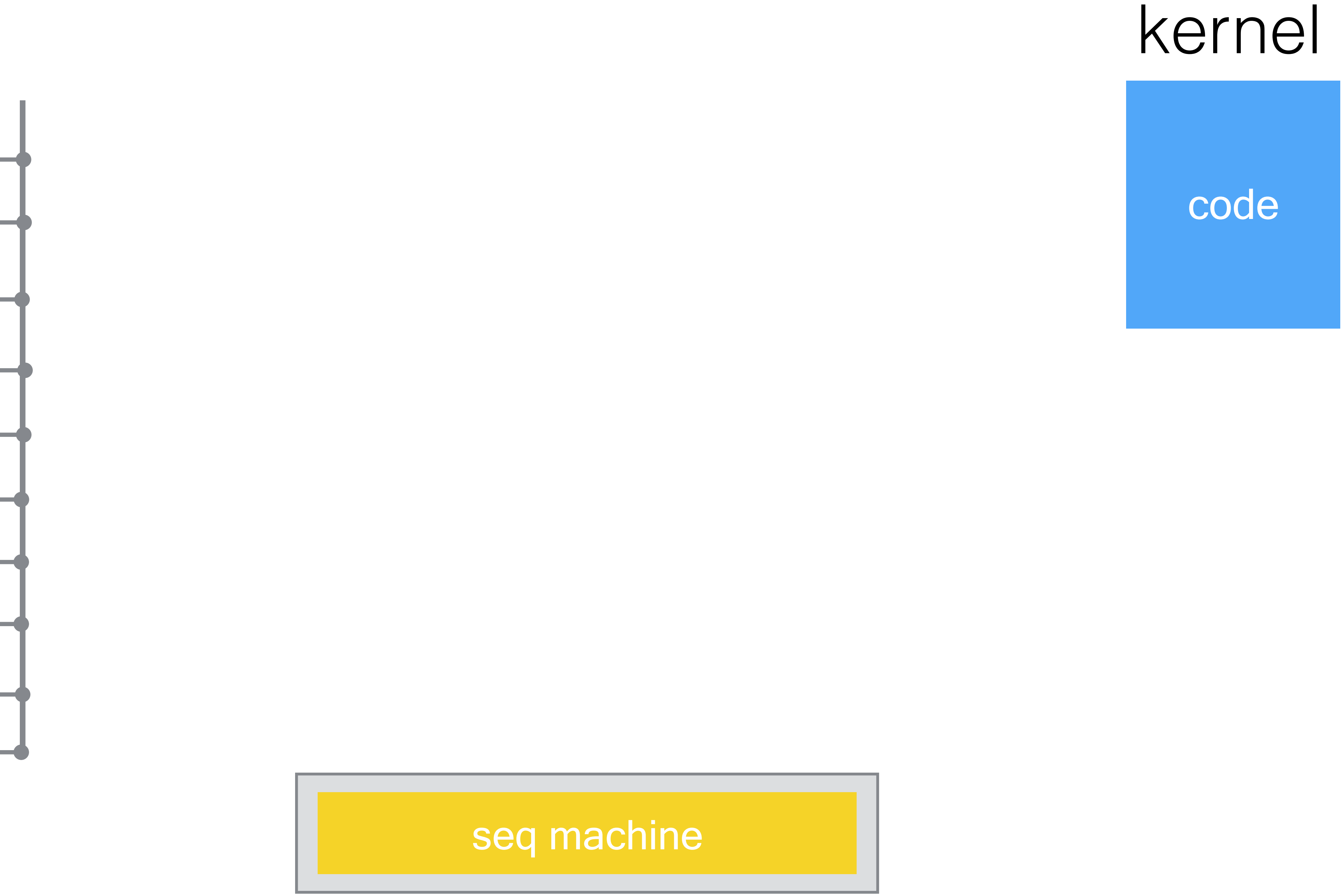

specification
implementation

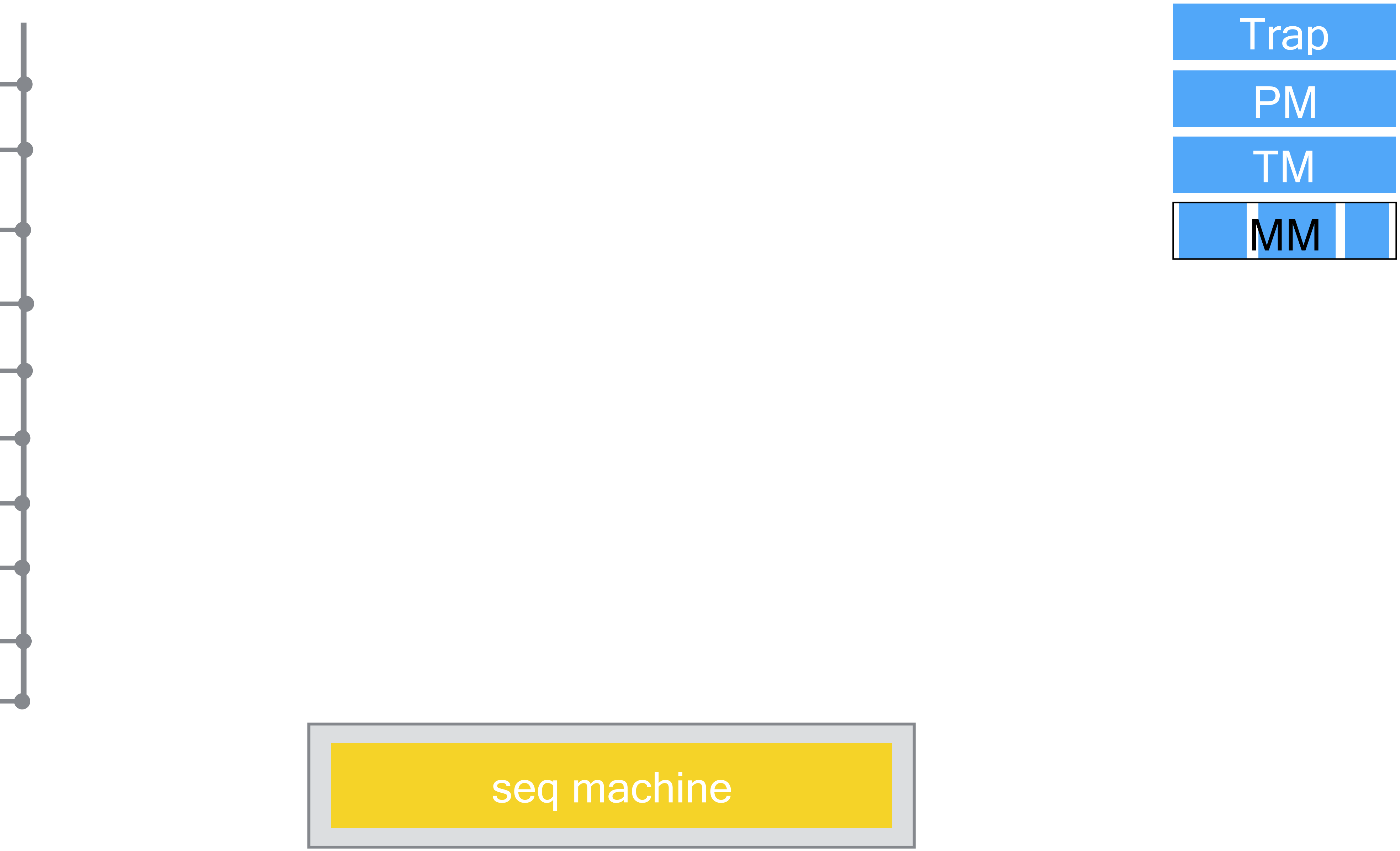
simulation proof

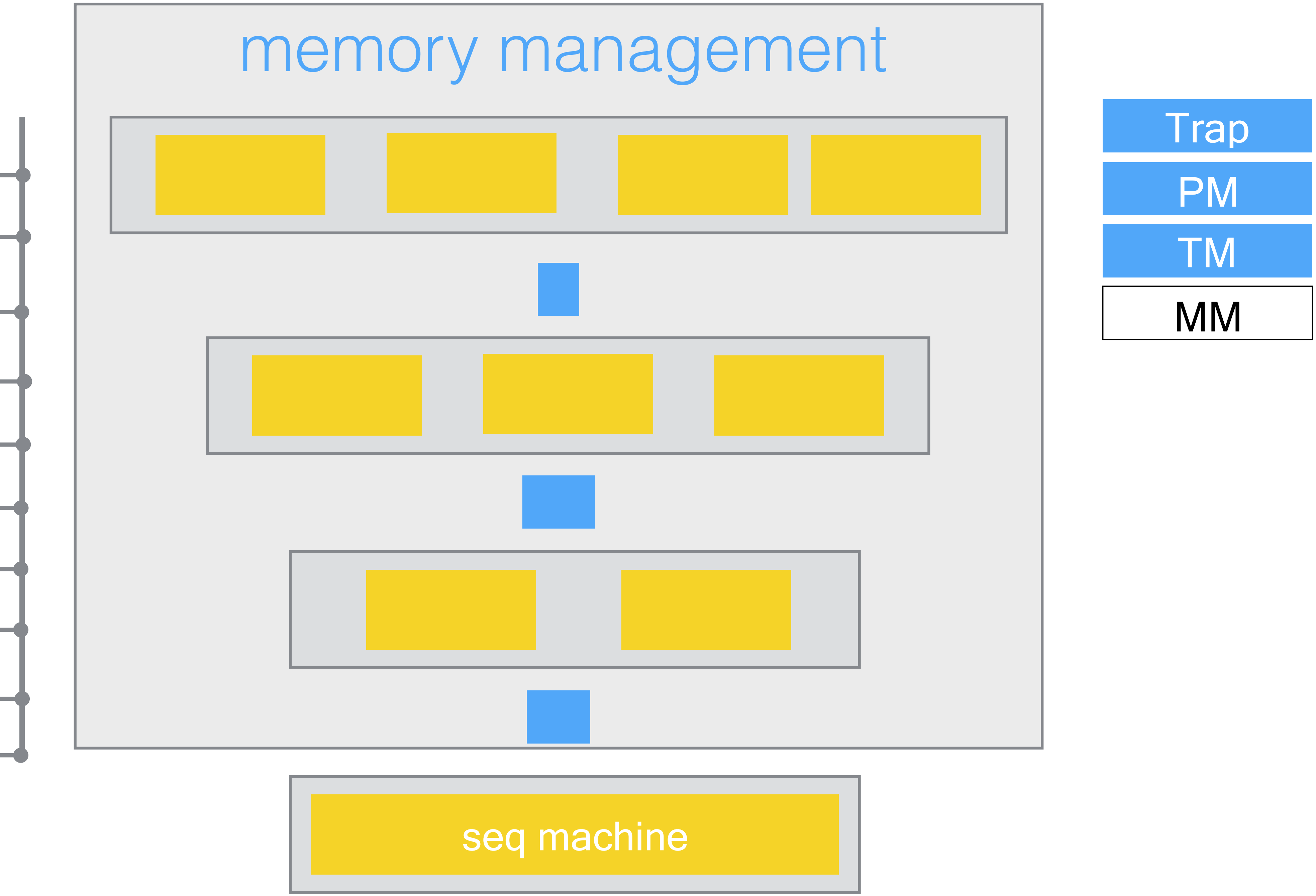


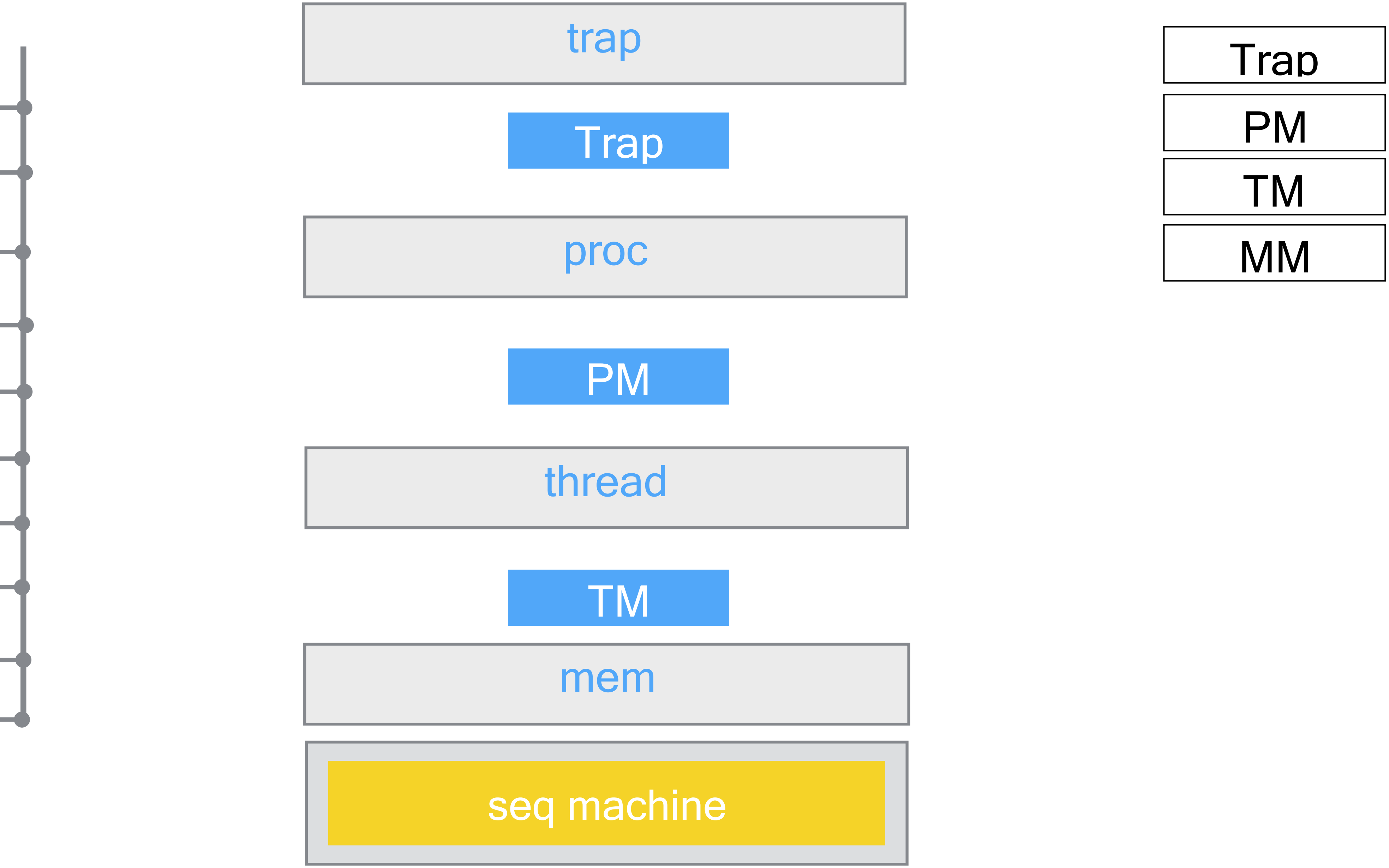
verify a sequential kernel

[POPL'15]

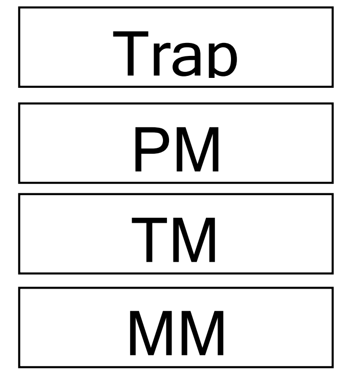

kernel
code
seq machine

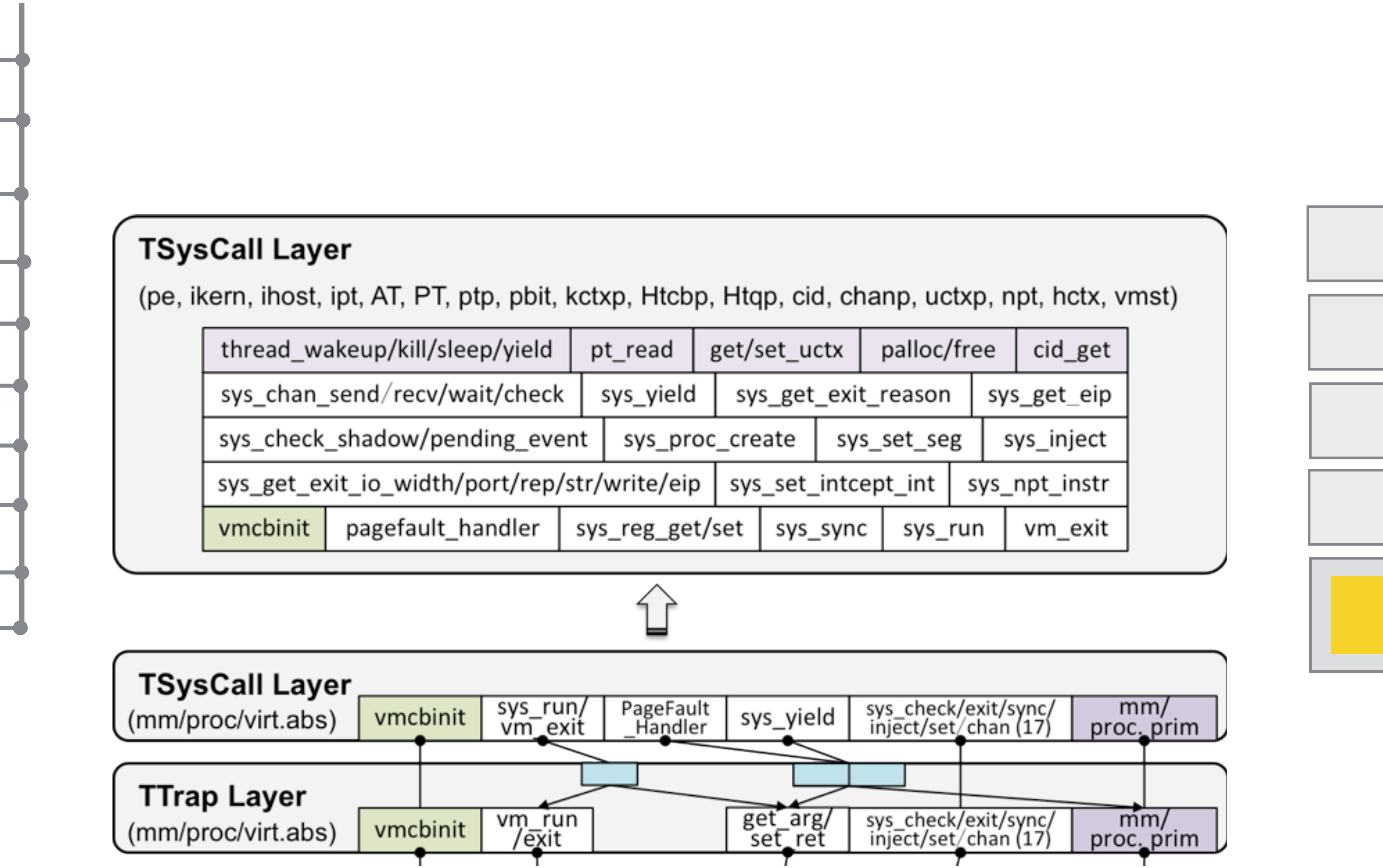

Trap
PM
TM
MM
seq machine

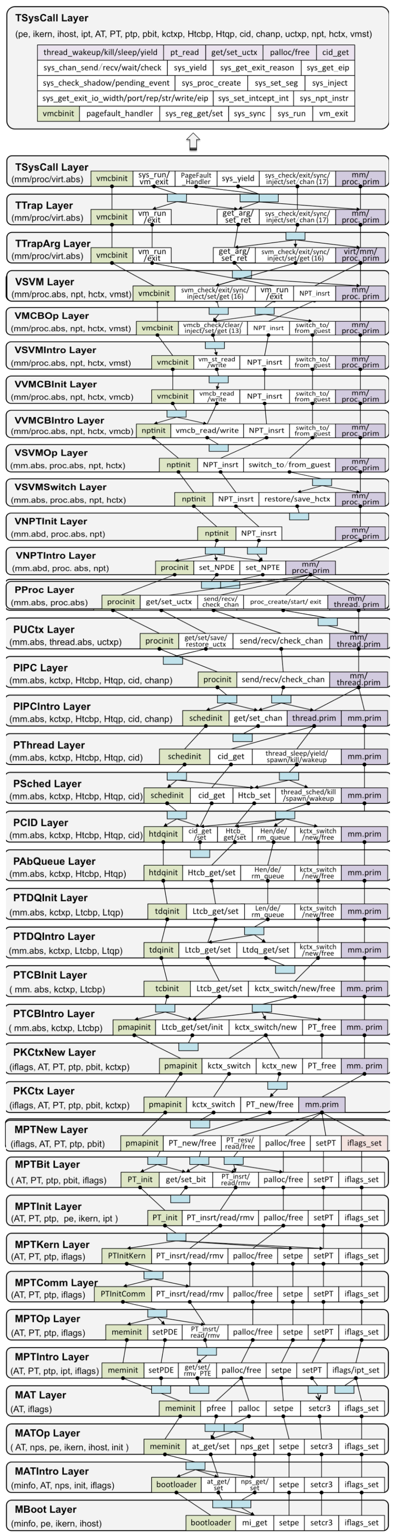

memory management
seq machine
Trap
PM
TM
MM


trap
Trap
proc
PM
thread
TM
mem
seq machine
Trap
PM
TM
MM

verified sequential kernel

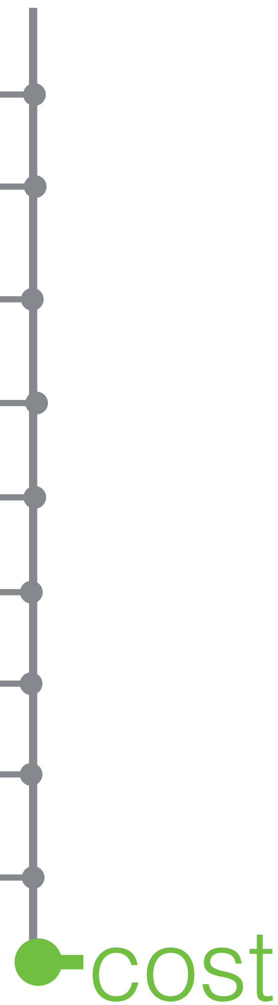



TSysCall Layer
(pe, ikern, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcbp, Htqp, cid, chanp, uctxp, npt, hctx, vmst)
thread_wakeup/kill/sleep/yield
pt_read
get/set_uctx
palloc/free
cid_get
sys_chan_send/recv/wait/check
sys_yield
sys_get_exit_reason
sys_get_eip
sys_check_shadow/pending_event
sys_proc_create
sys_set_seg
sys_inject
sys_get_exit_io_width/port/rep/str/write/eip
sys_set_intcept_int
sys_npt_instr
vmcbinit
pagefault_handler
sys_reg_get/set
sys_sync
sys_run
vm_exit
TSysCall Layer
(mm/proc/virt.abs)
vmcbinit
sys_run/ vm_exit
PageFault _Handler
sys_yield
sys_check/exit/sync/ inject/set/chan (17)
mm/ proc. prim
TTrap Layer
(mm/proc/virt.abs)
vmcbinit
vm_run /exit
get_arg/ set_ret
sys_check/exit/sync/ inject/set/chan (17)
mm/ proc. prim

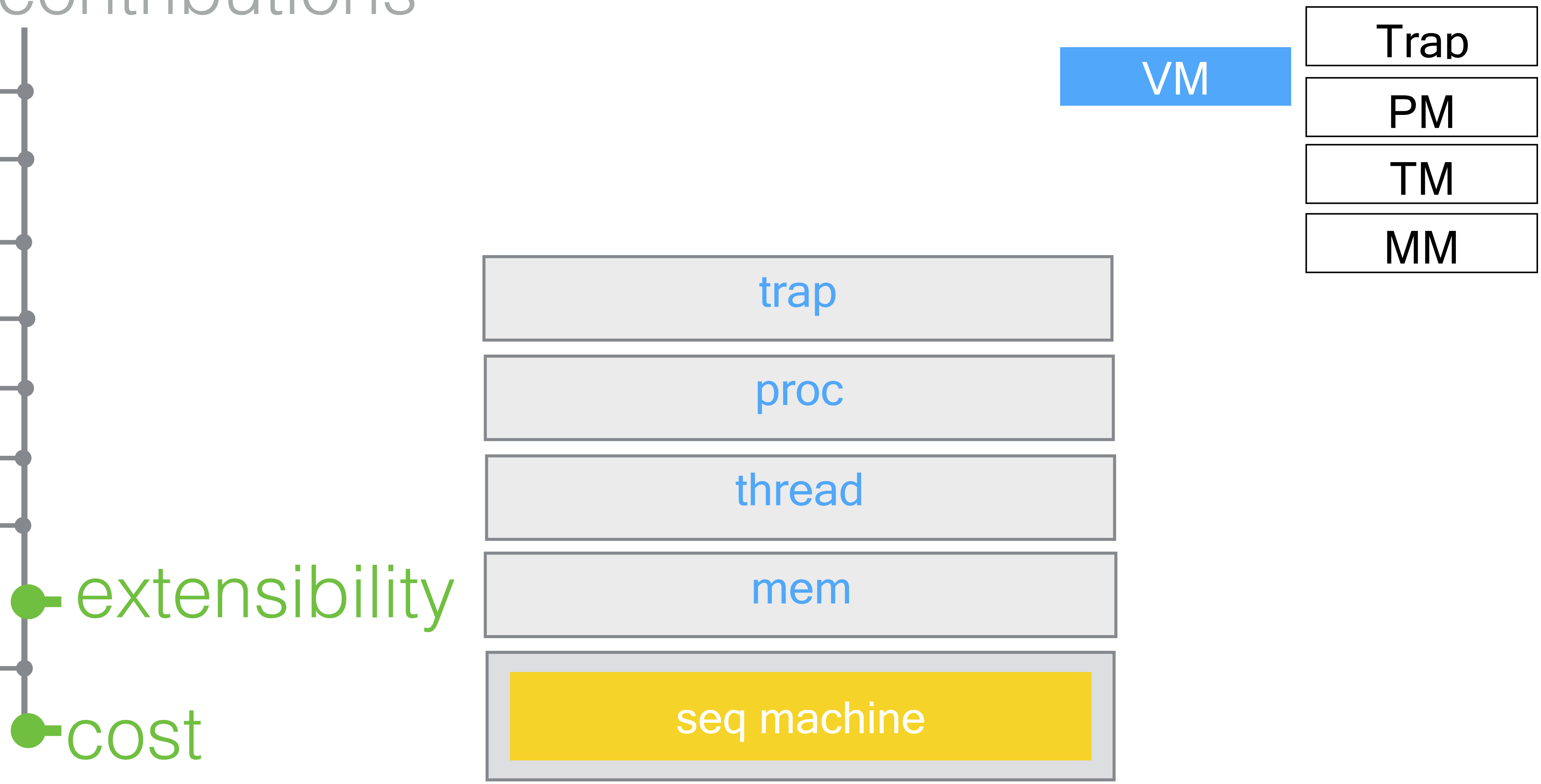


1 person year

(cost for tool construction excluded)

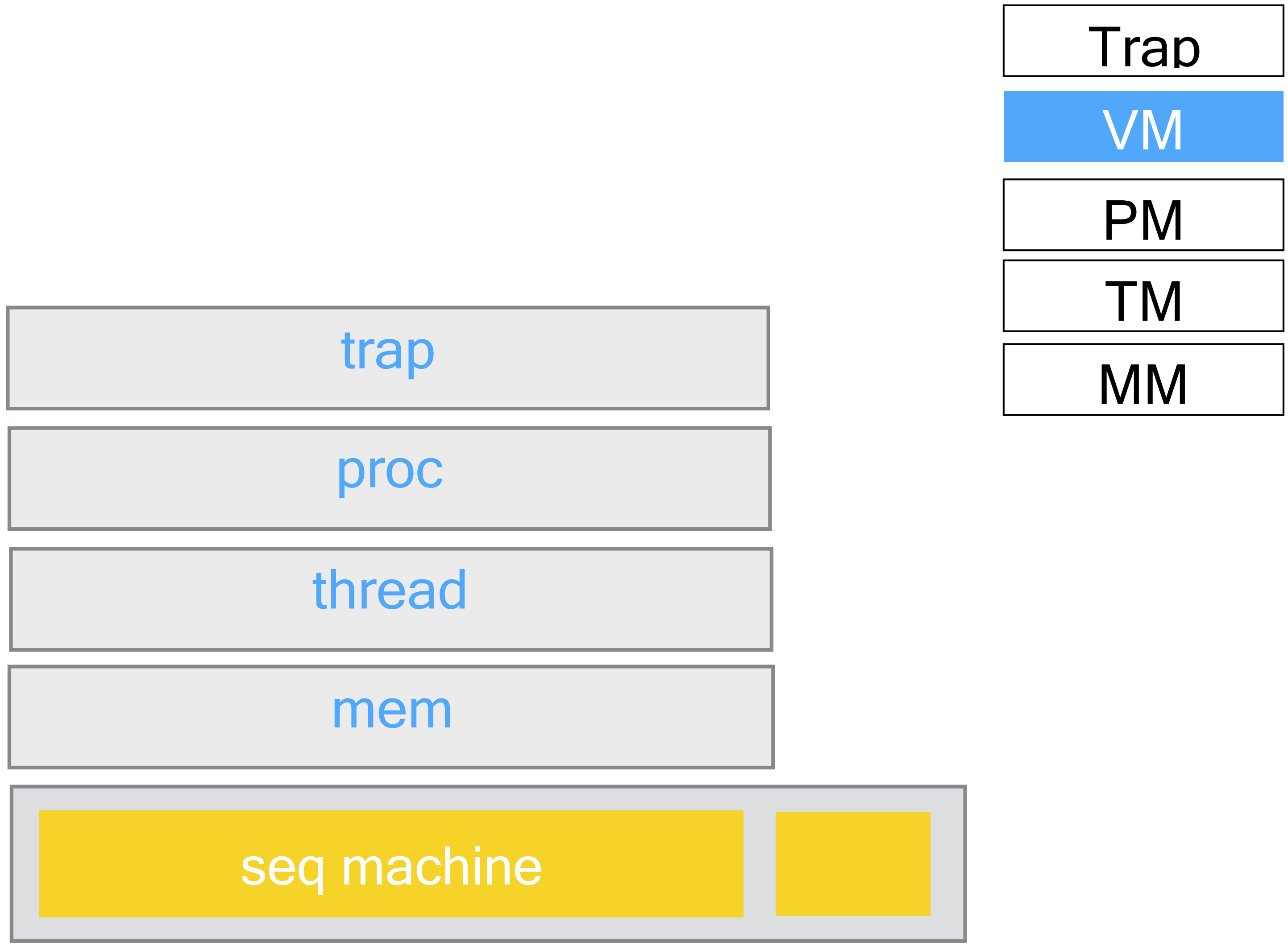


contributions



contributions

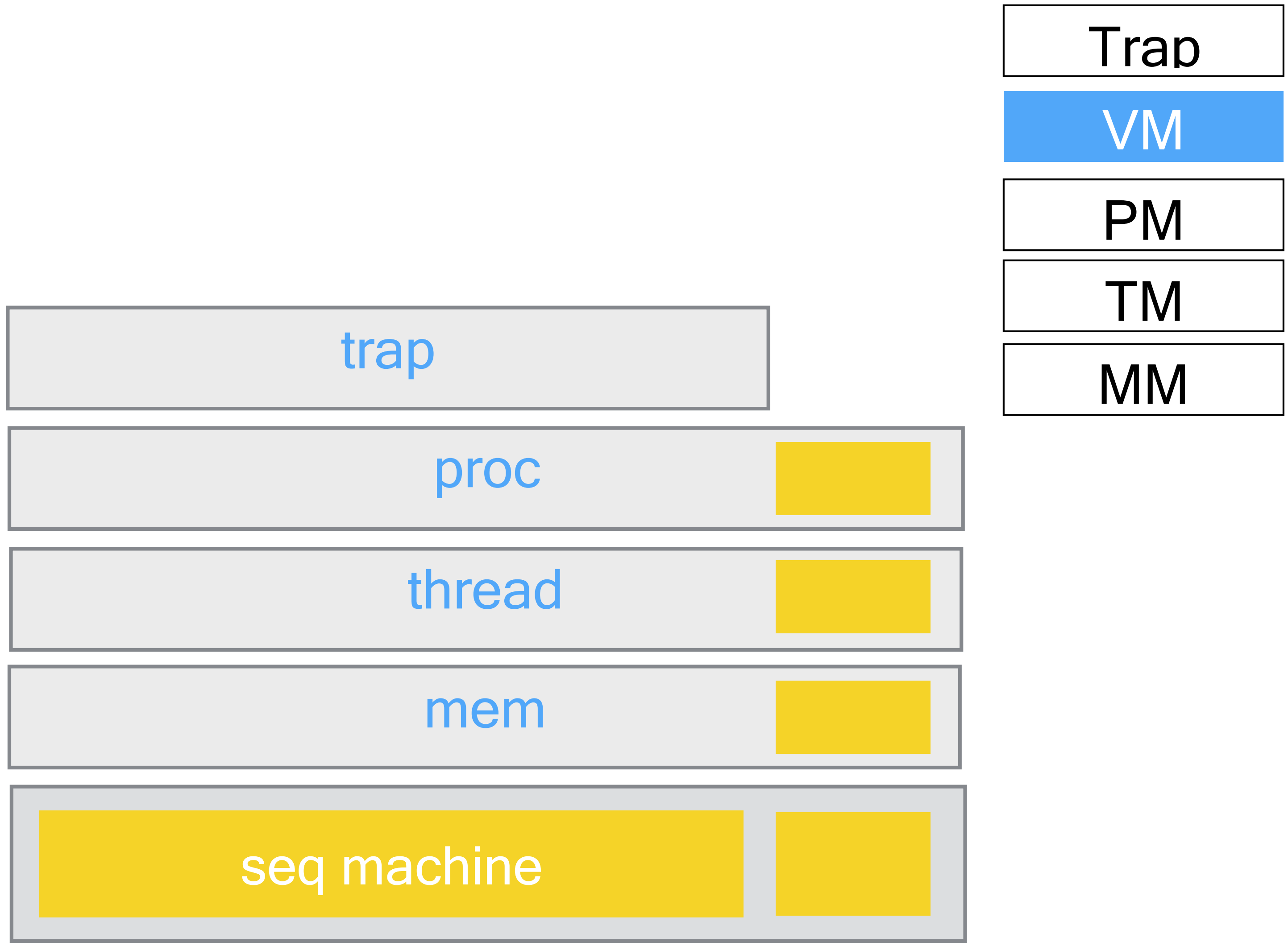
- extensibility
- cost



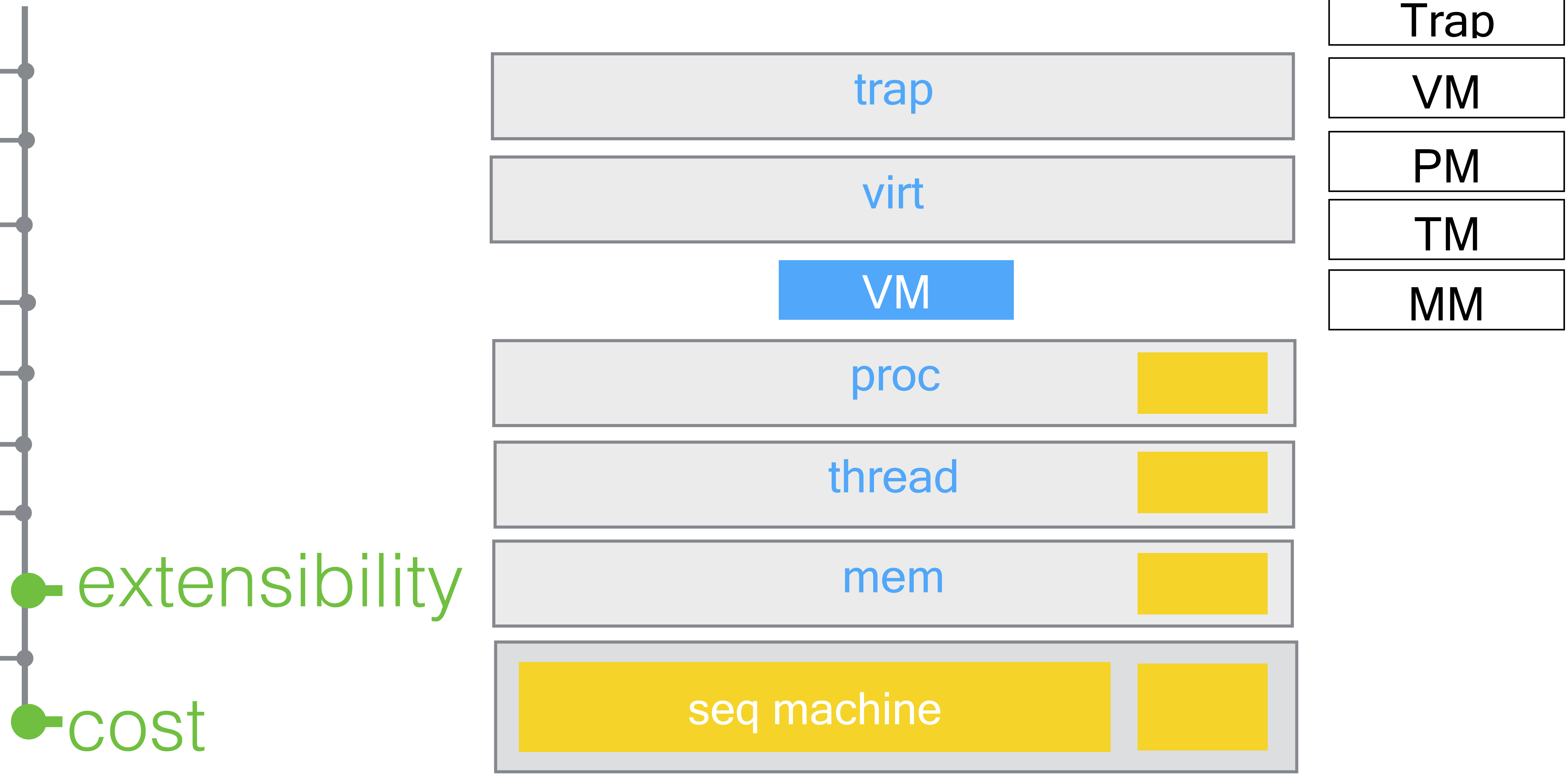
contributions

extensibility

cost

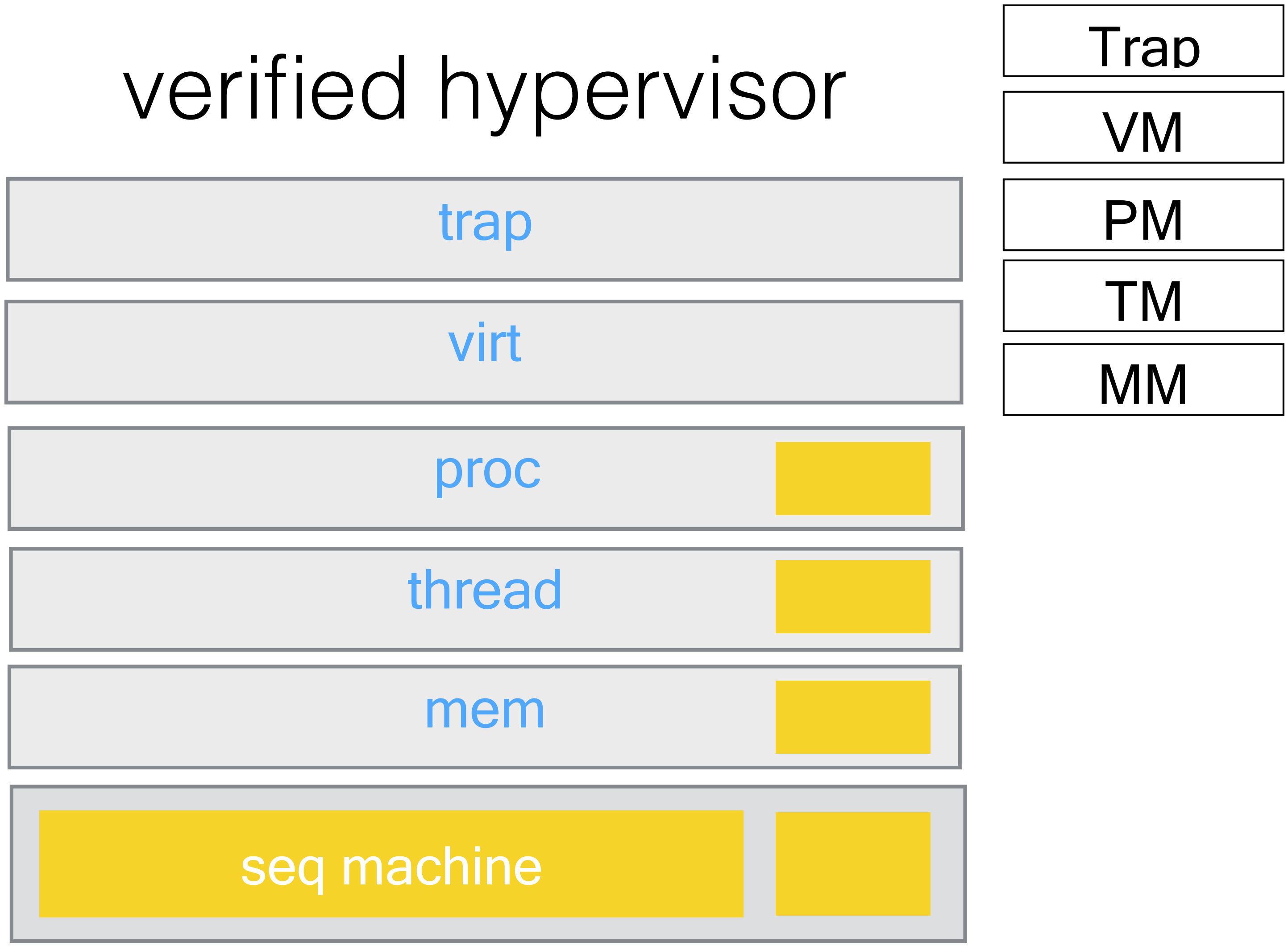


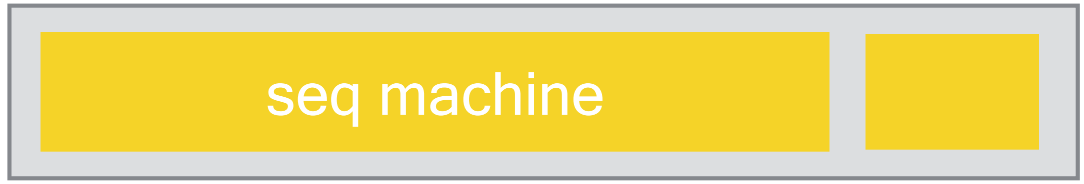
contributions



contributions

verified hypervisor



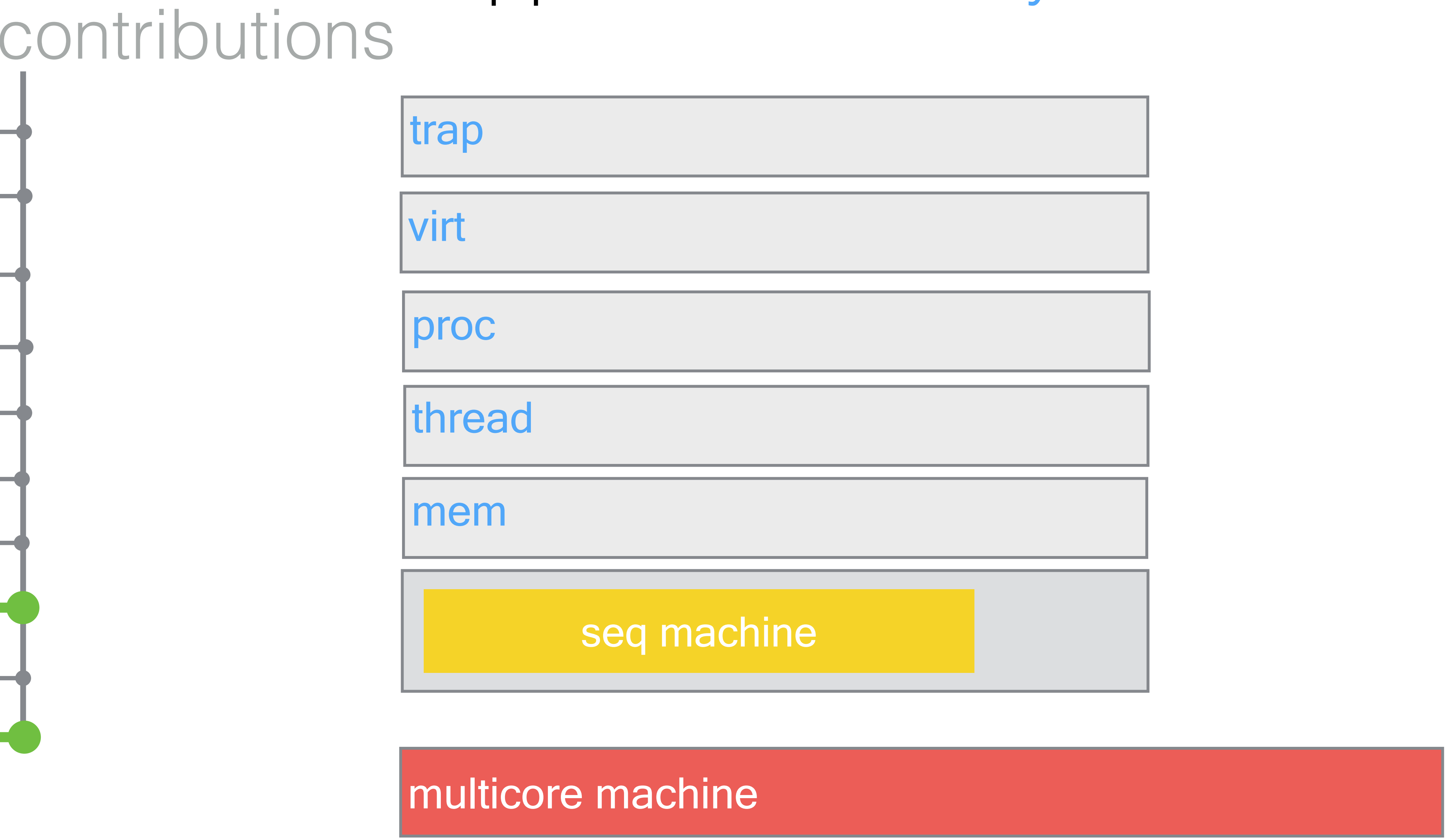


contributions

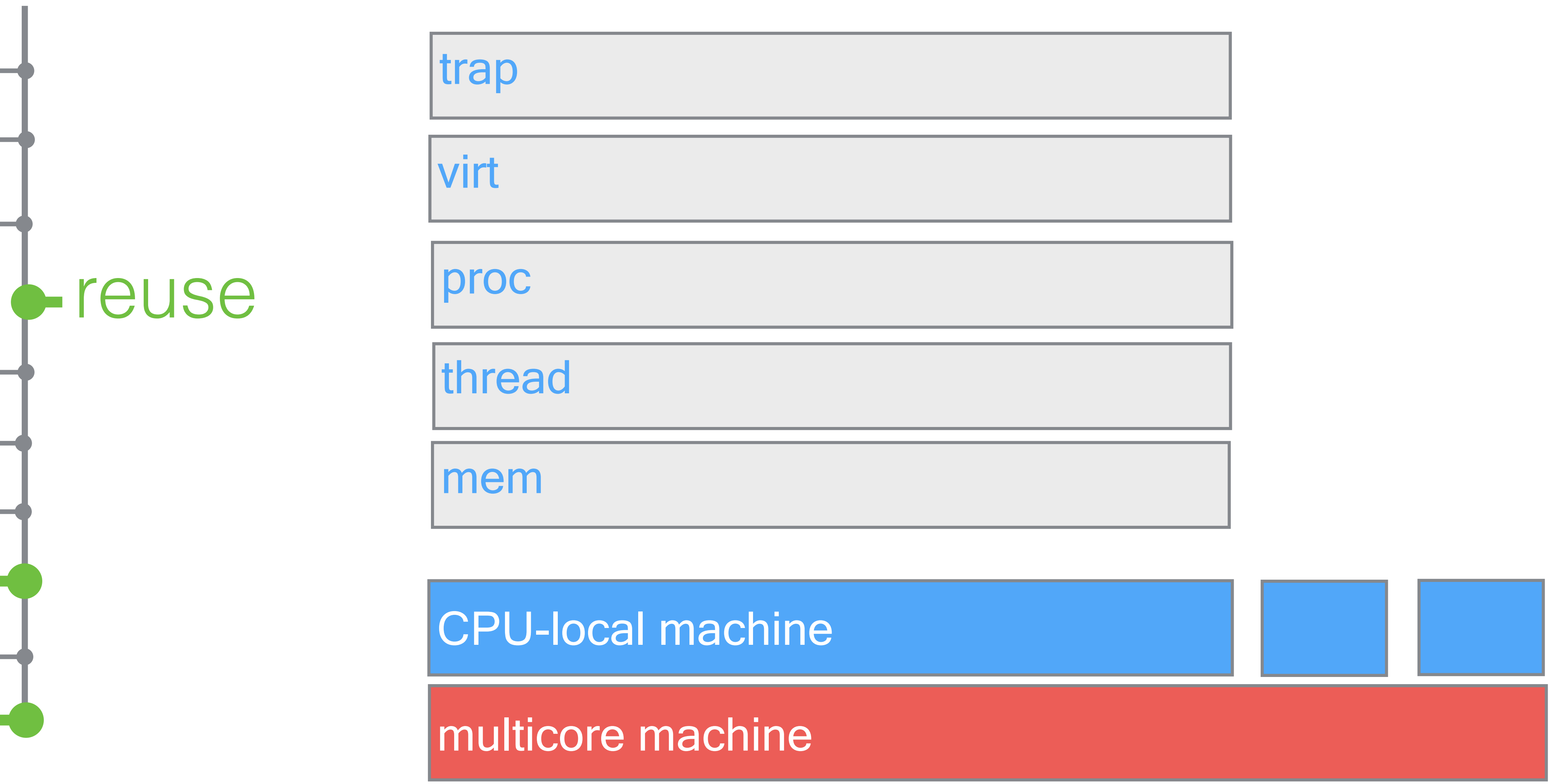
extensibility is the key to support

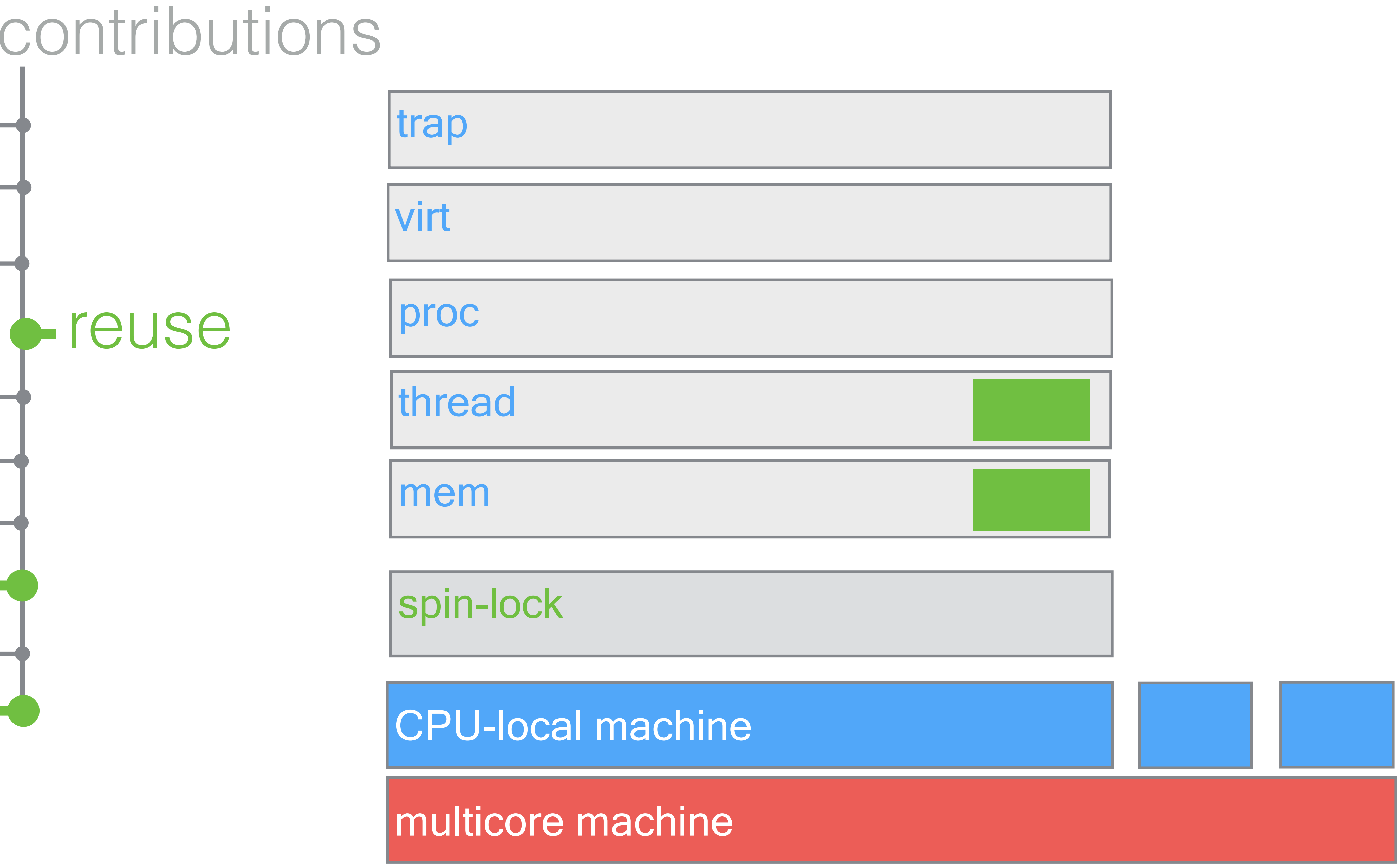
concurrency

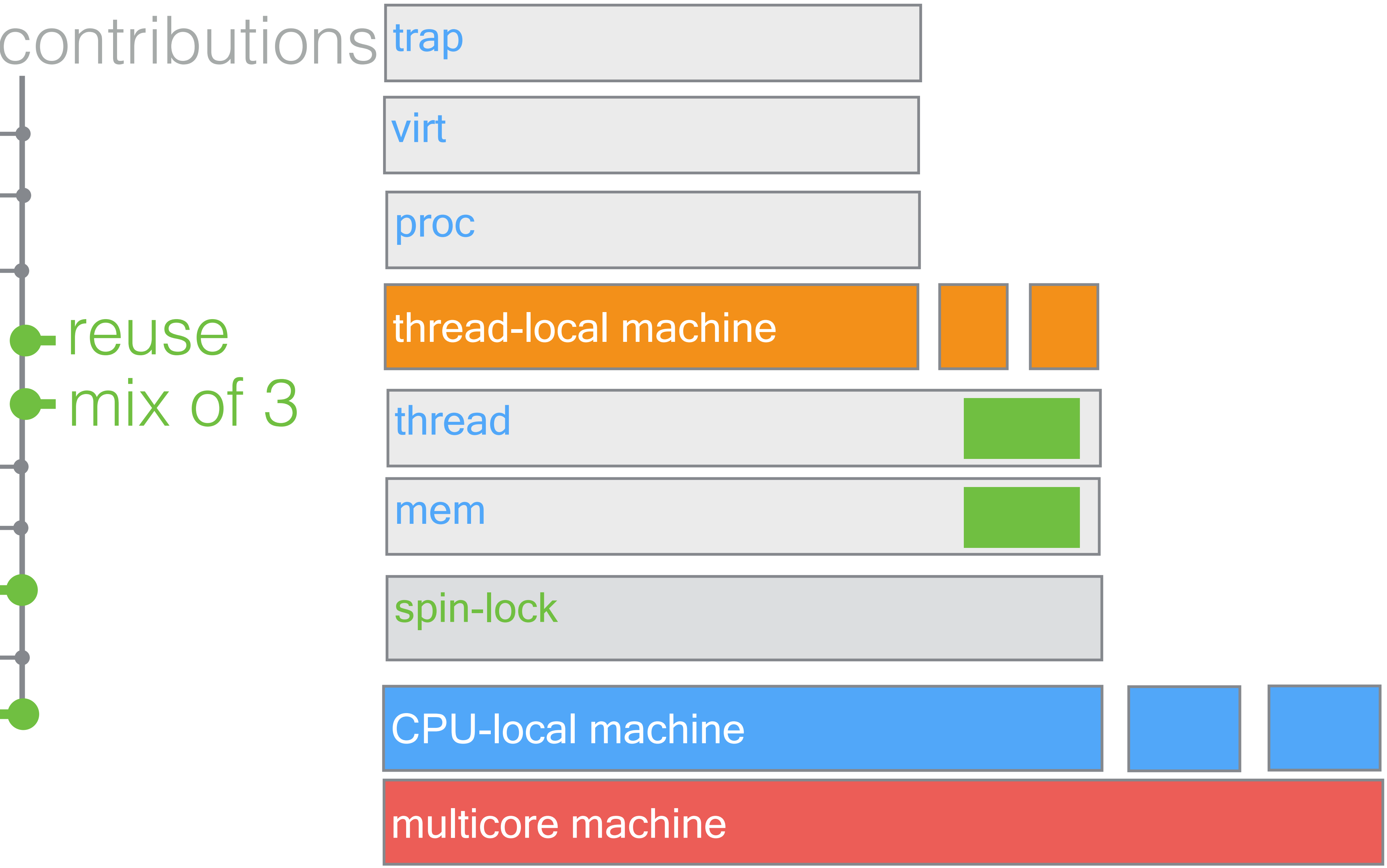
support concurrency

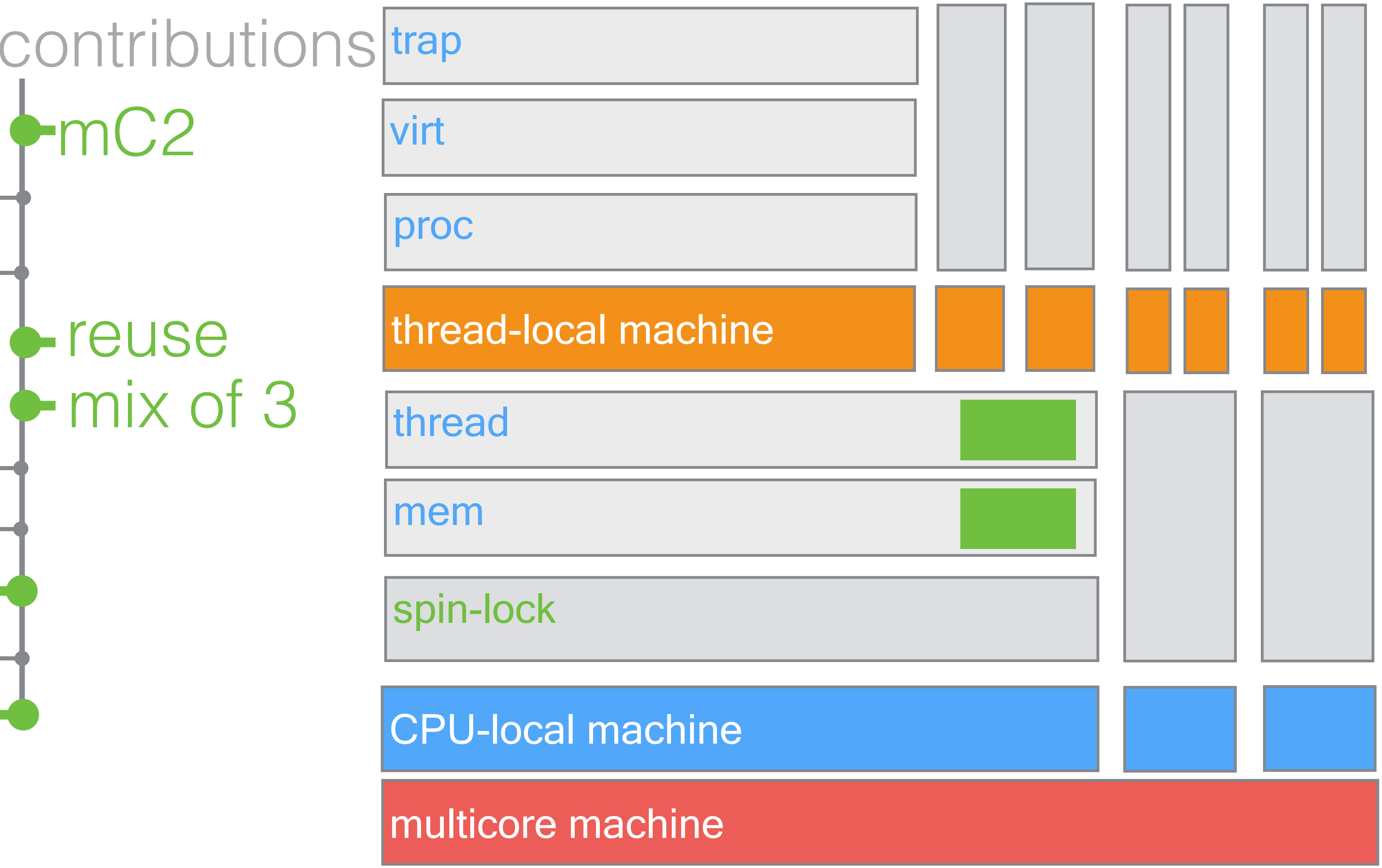


contributions

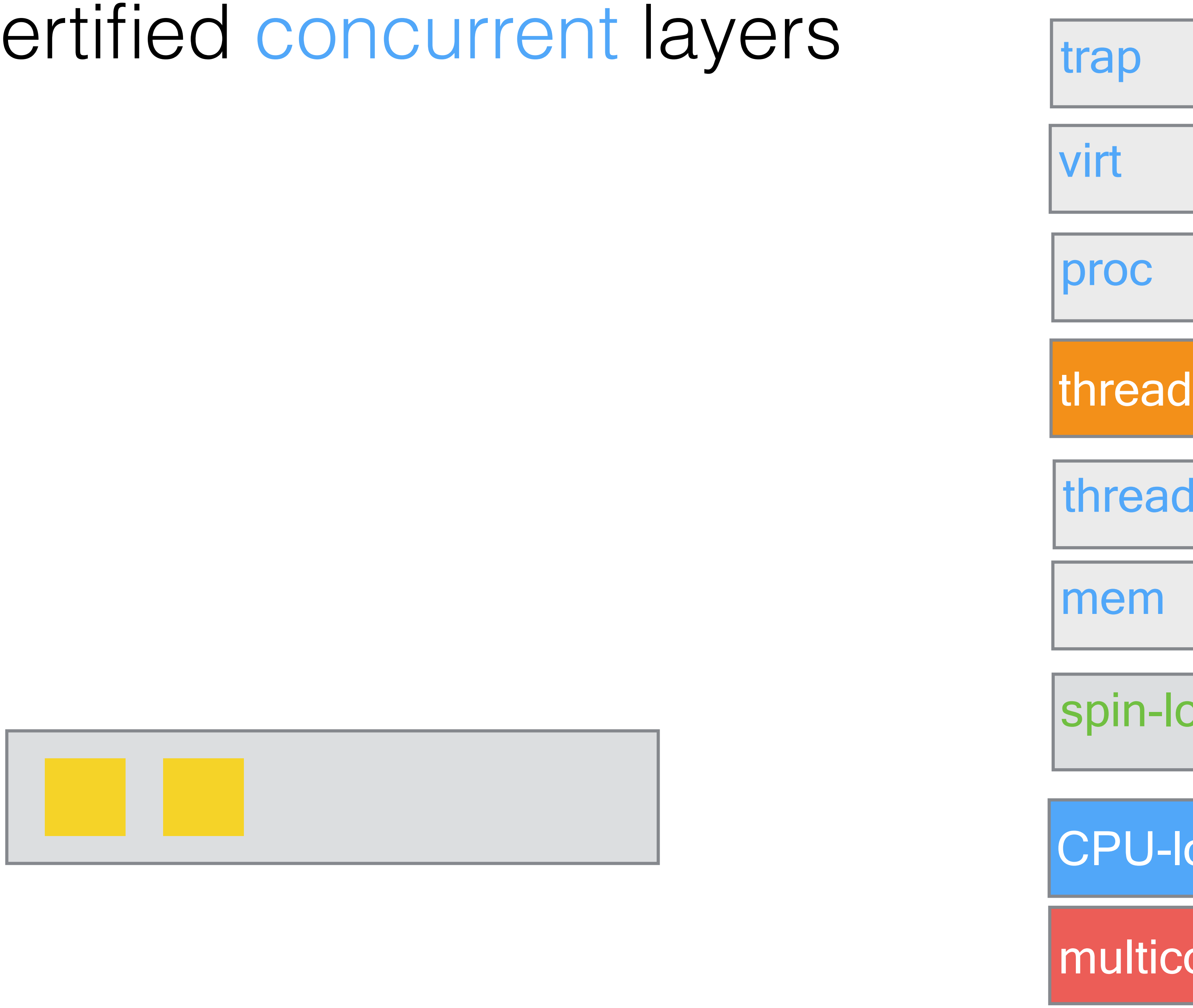



contributions
reuse
trap
virt
proc
thread
mem
spin-lock
CPU-local machine
multicore machine


contributions
reuse
mix of 3
trap
virt
proc
thread-local machine
thread
mem
spin-lock
CPU-local machine
multicore machine

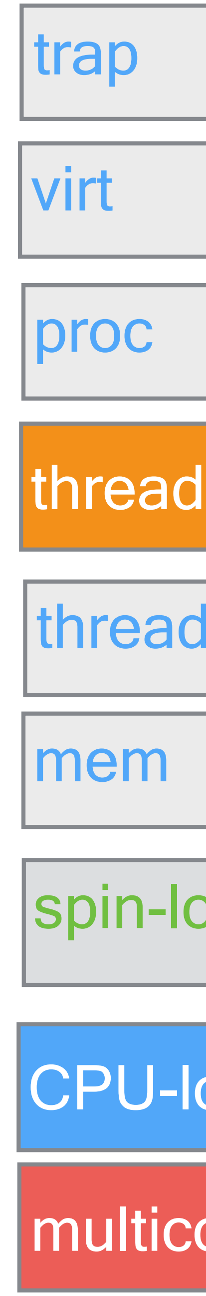

contributions
mC2
reuse
mix of 3
trap
virt
proc
thread-local machine
thread
mem
spin-lock
CPU-local machine
multicore machine

certified concurrent layers

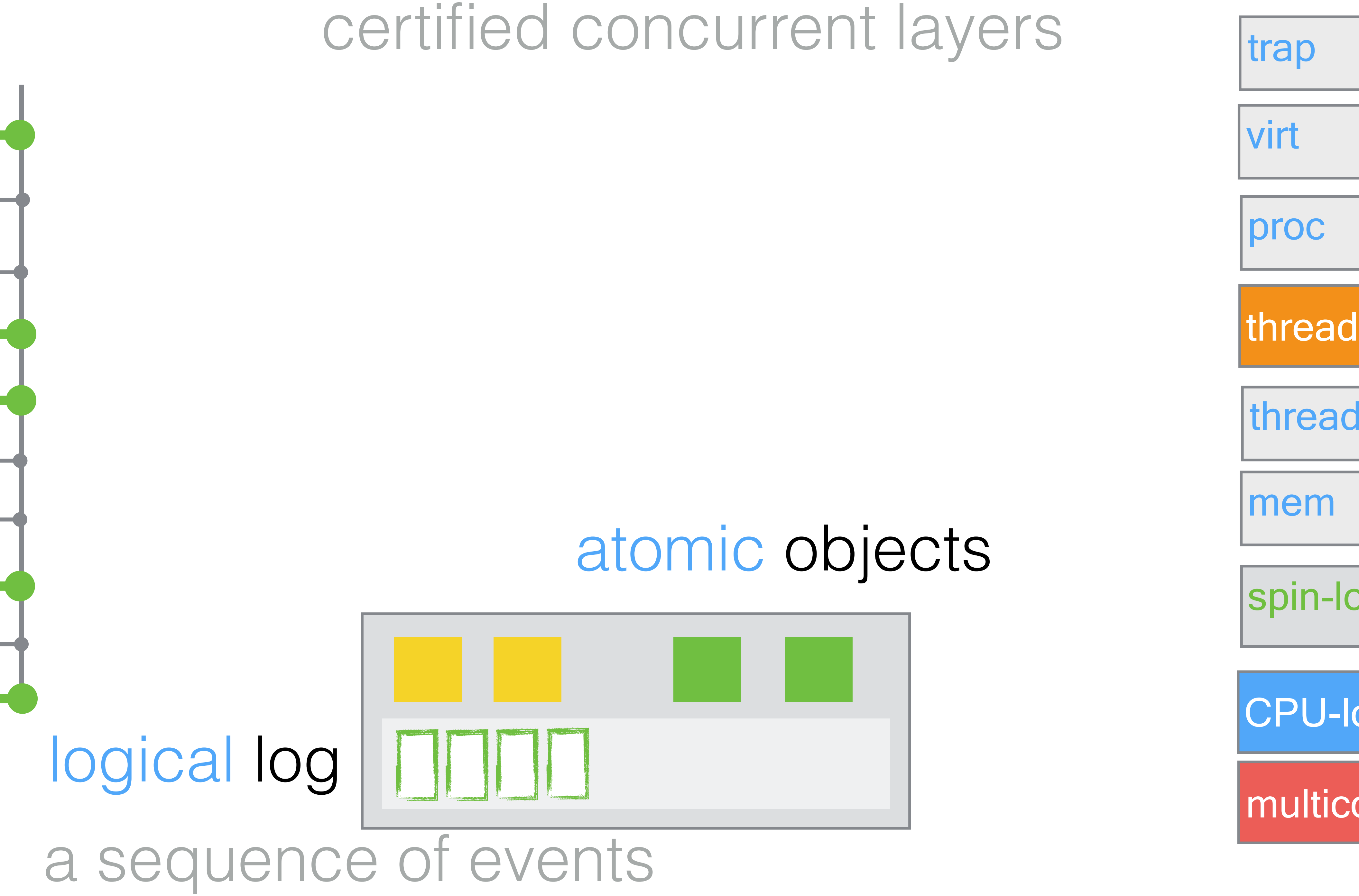


certified concurrent layers

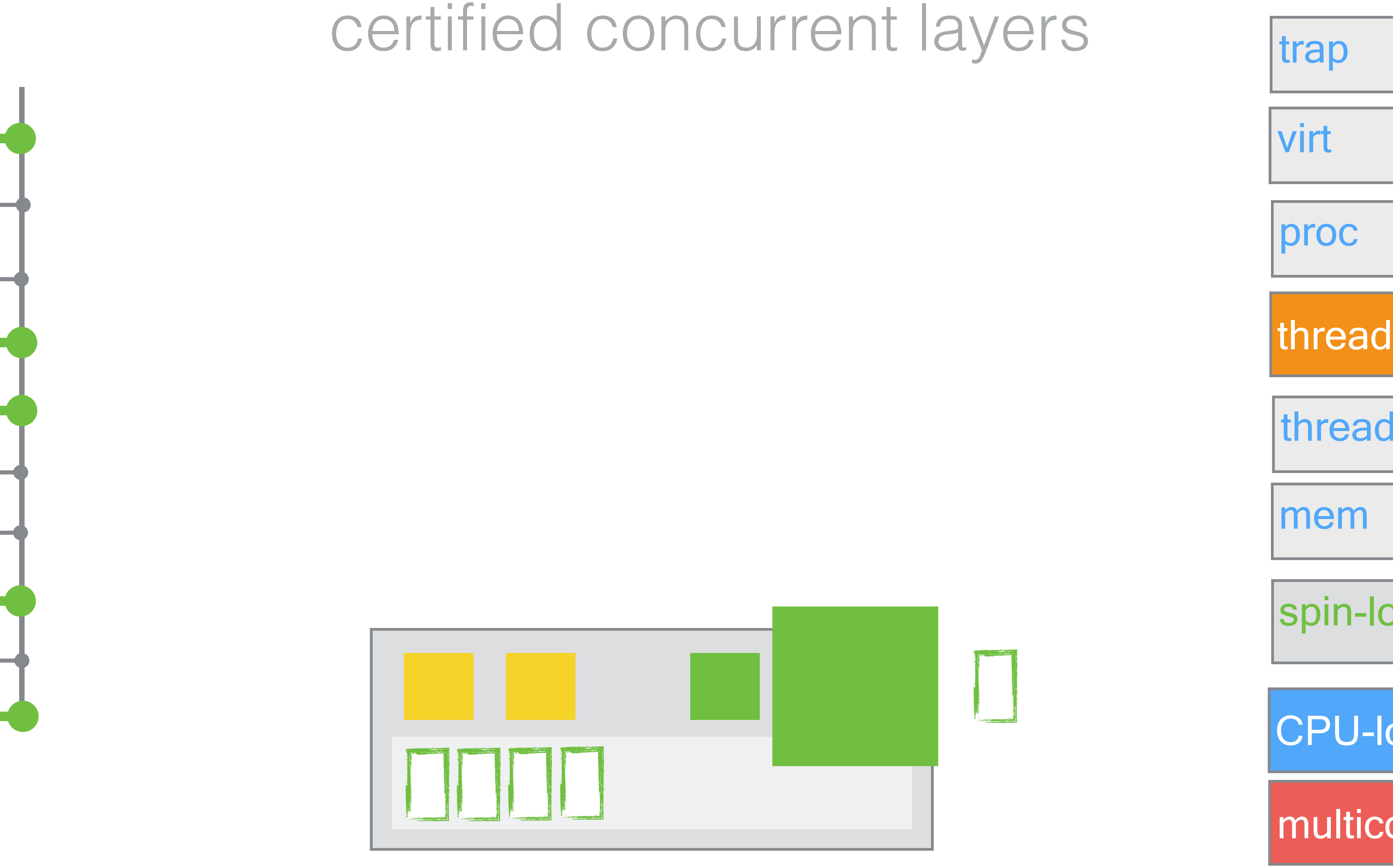
local objects



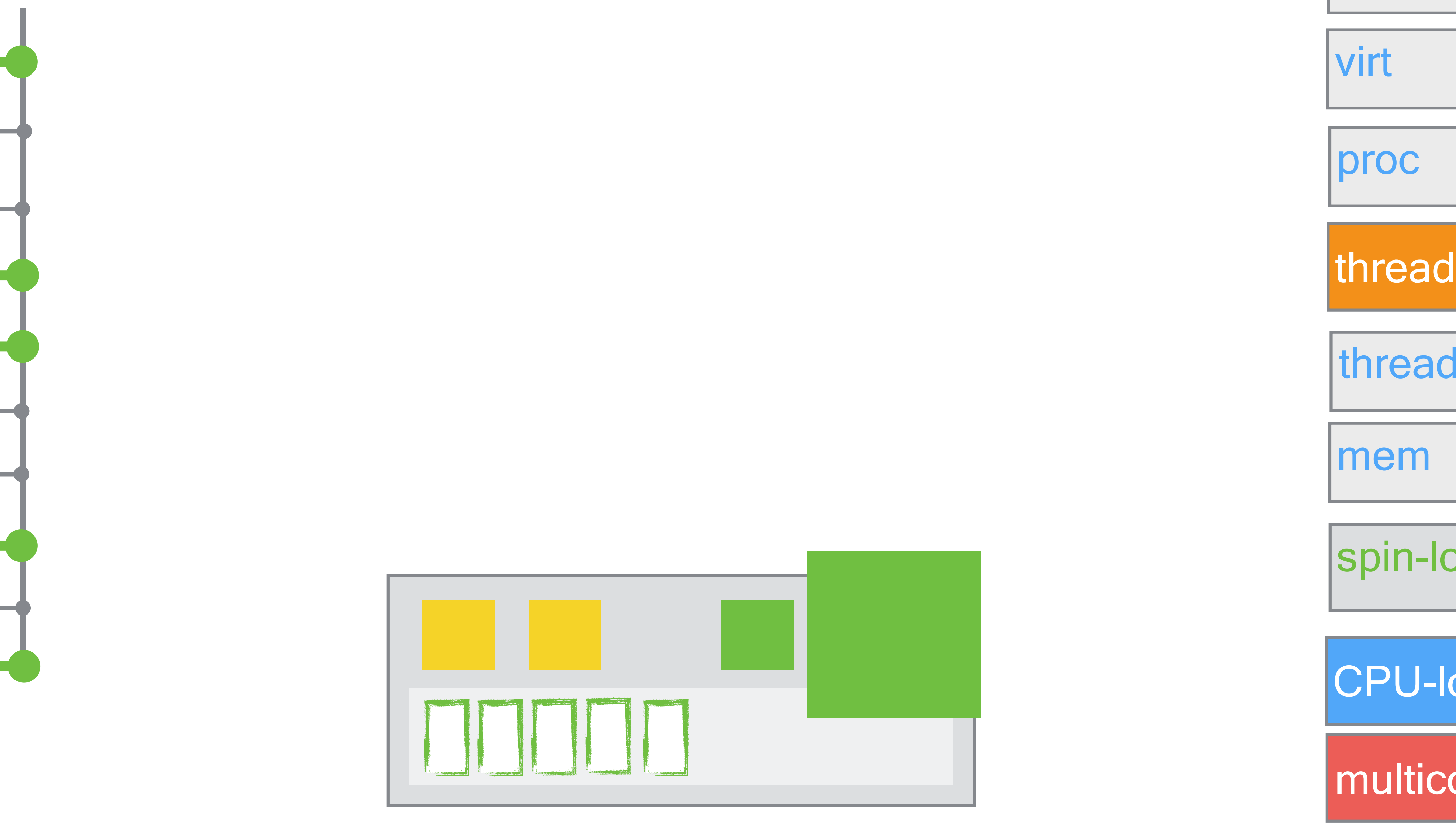
certified concurrent layers



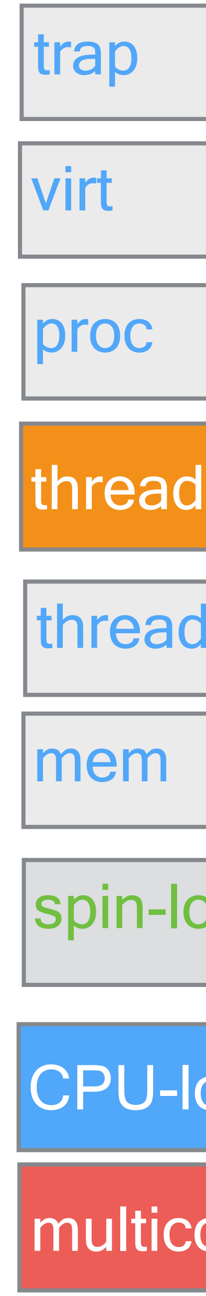
certified concurrent layers

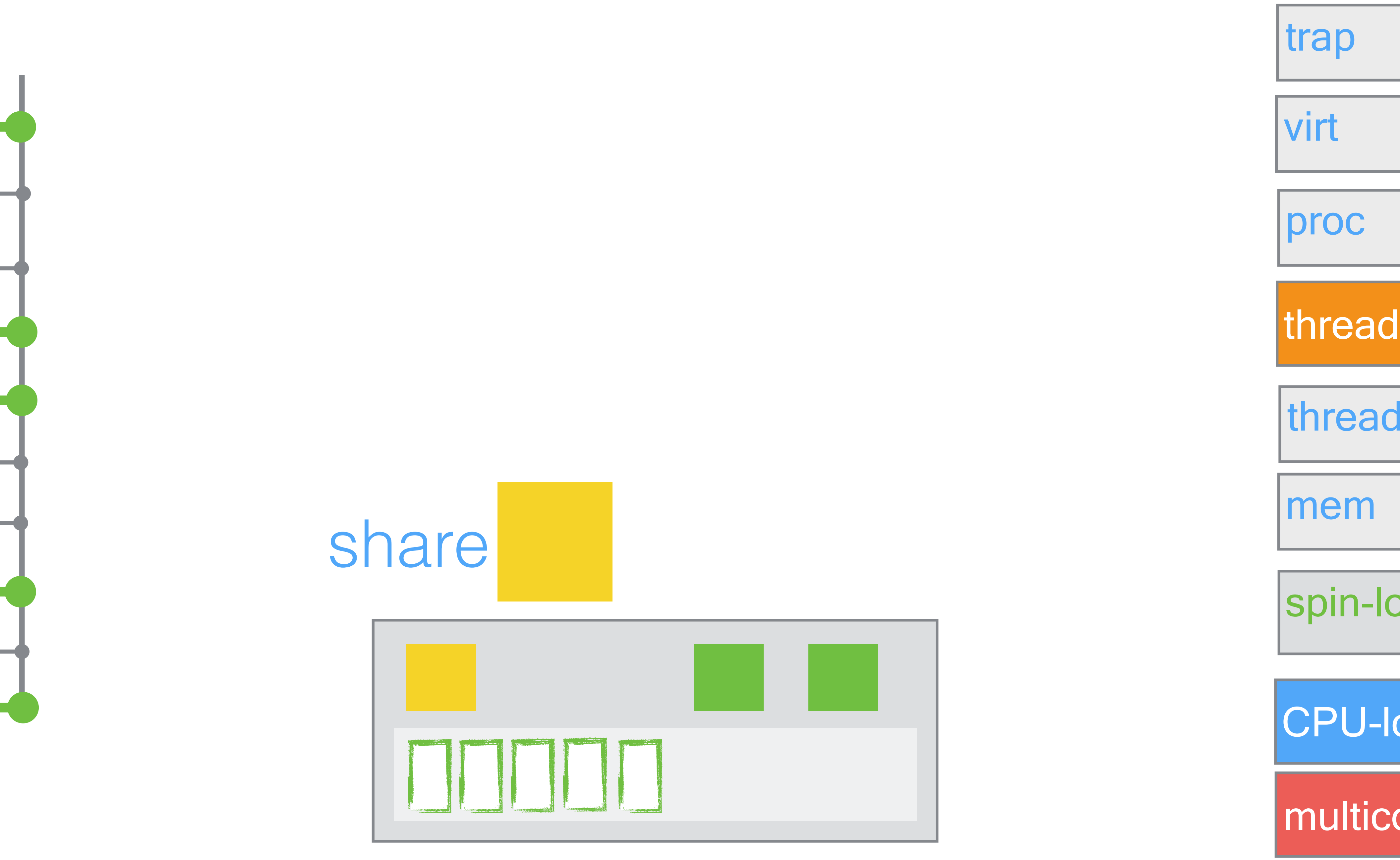


certified concurrent layers

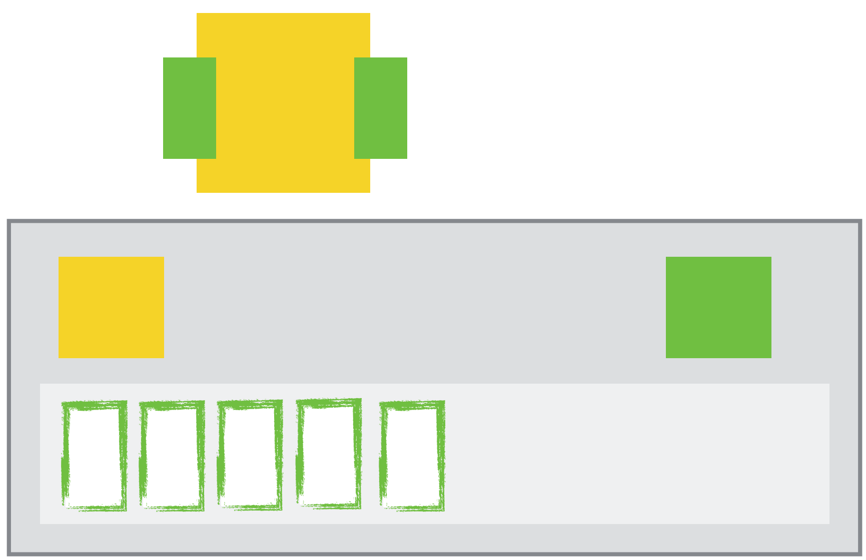


certified concurrent layers




trap
virt
proc
thread
thread
mem
share

fine-grained lock



trap

virt

proc

thread

thread

mem

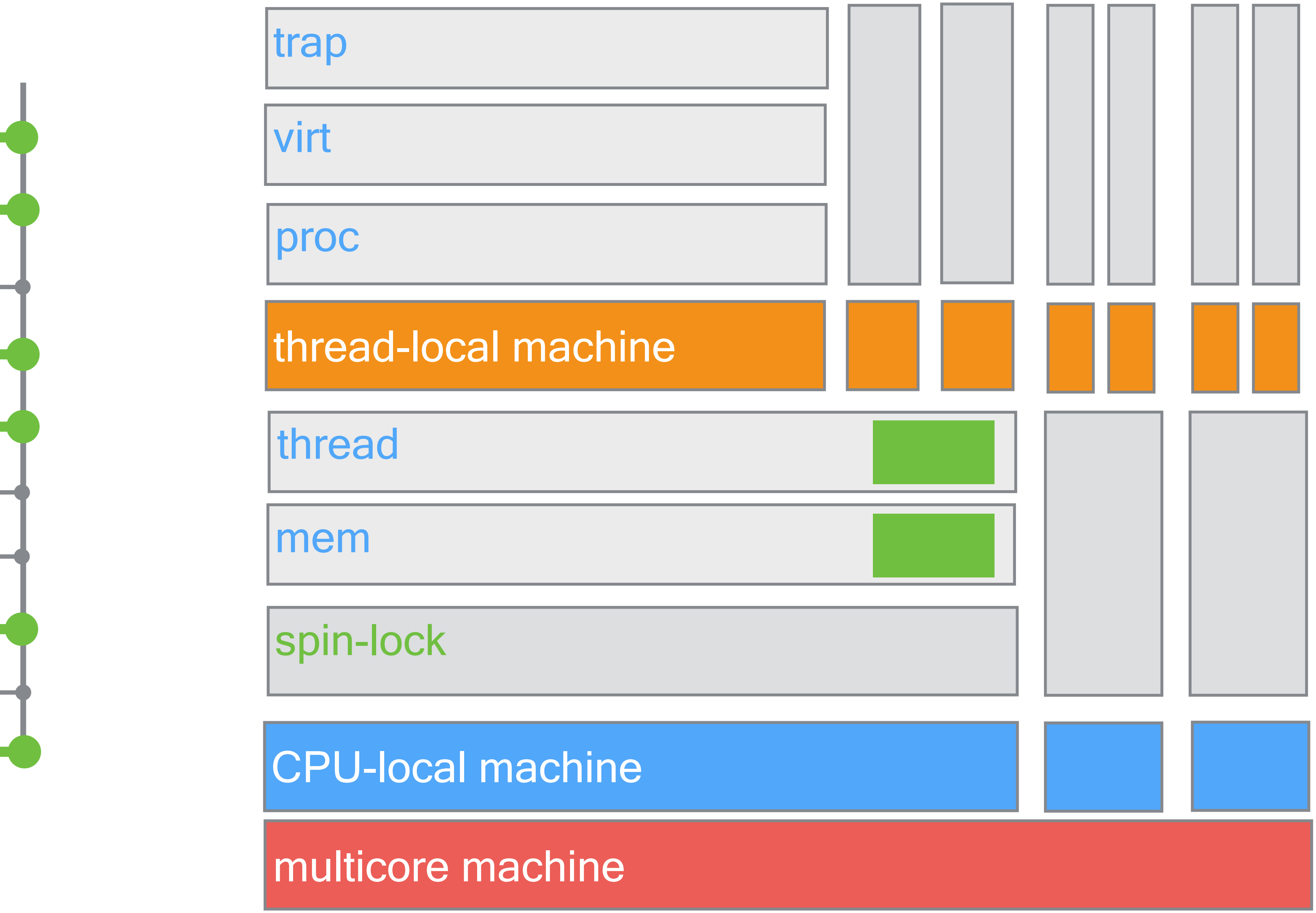
spin-lo

CPU-lo

multic

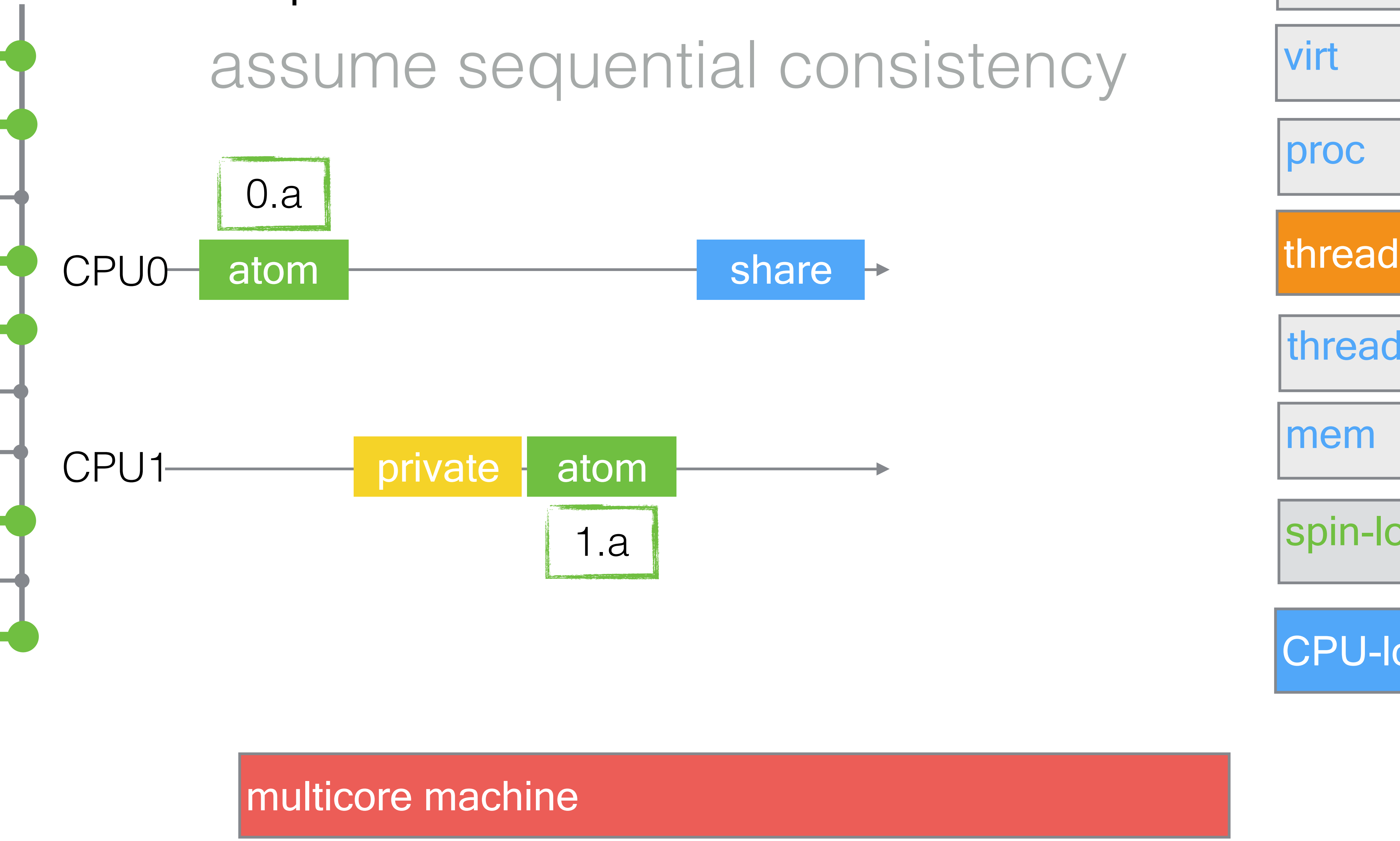
fine-grained lock

trap
virt
proc
thread
thread
mem

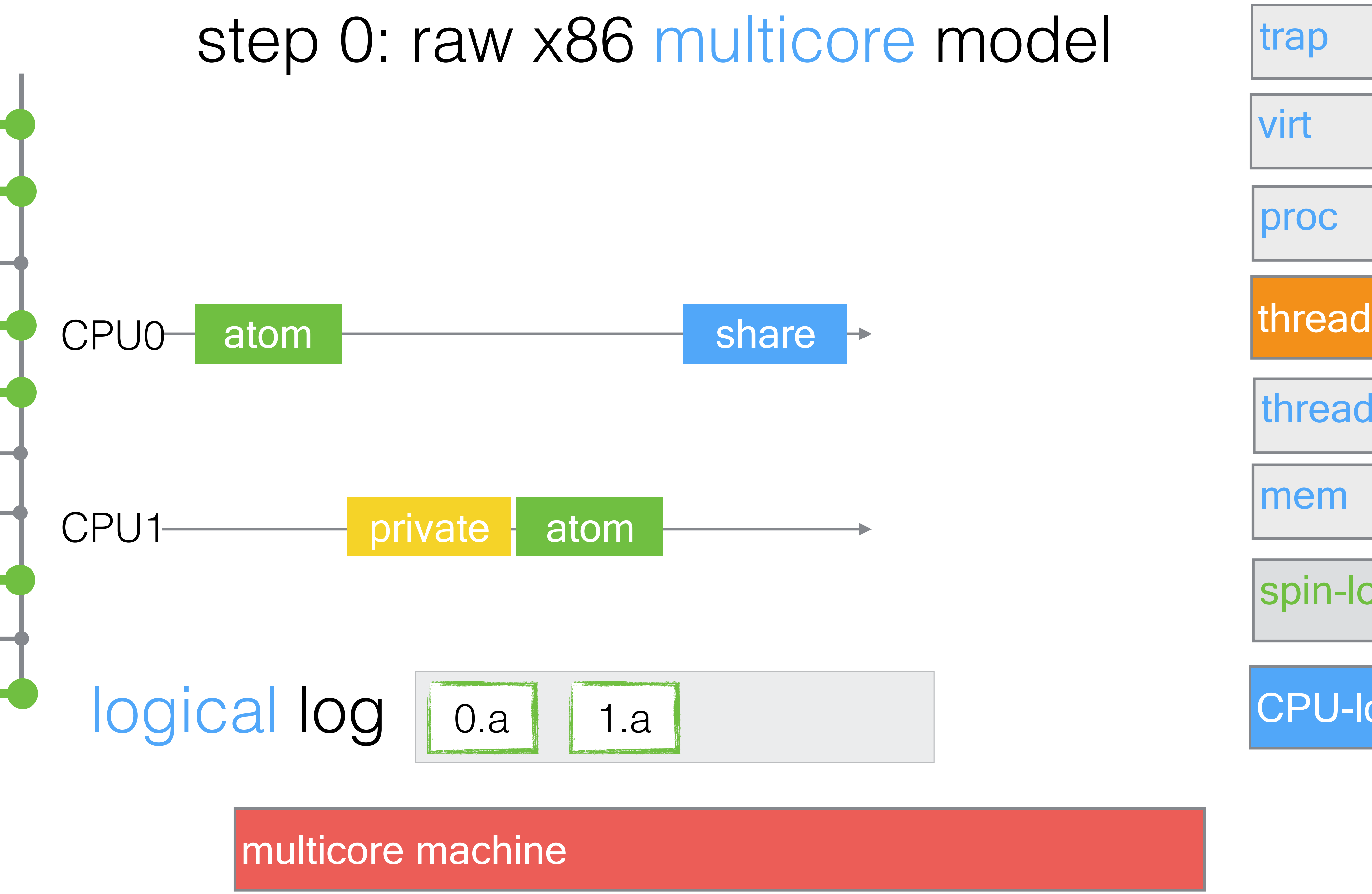

trap
virt
proc
thread-local machine
thread
mem
spin-lock
CPU-local machine
multicore machine

step 0: raw x86 multicore model

assume sequential consistency

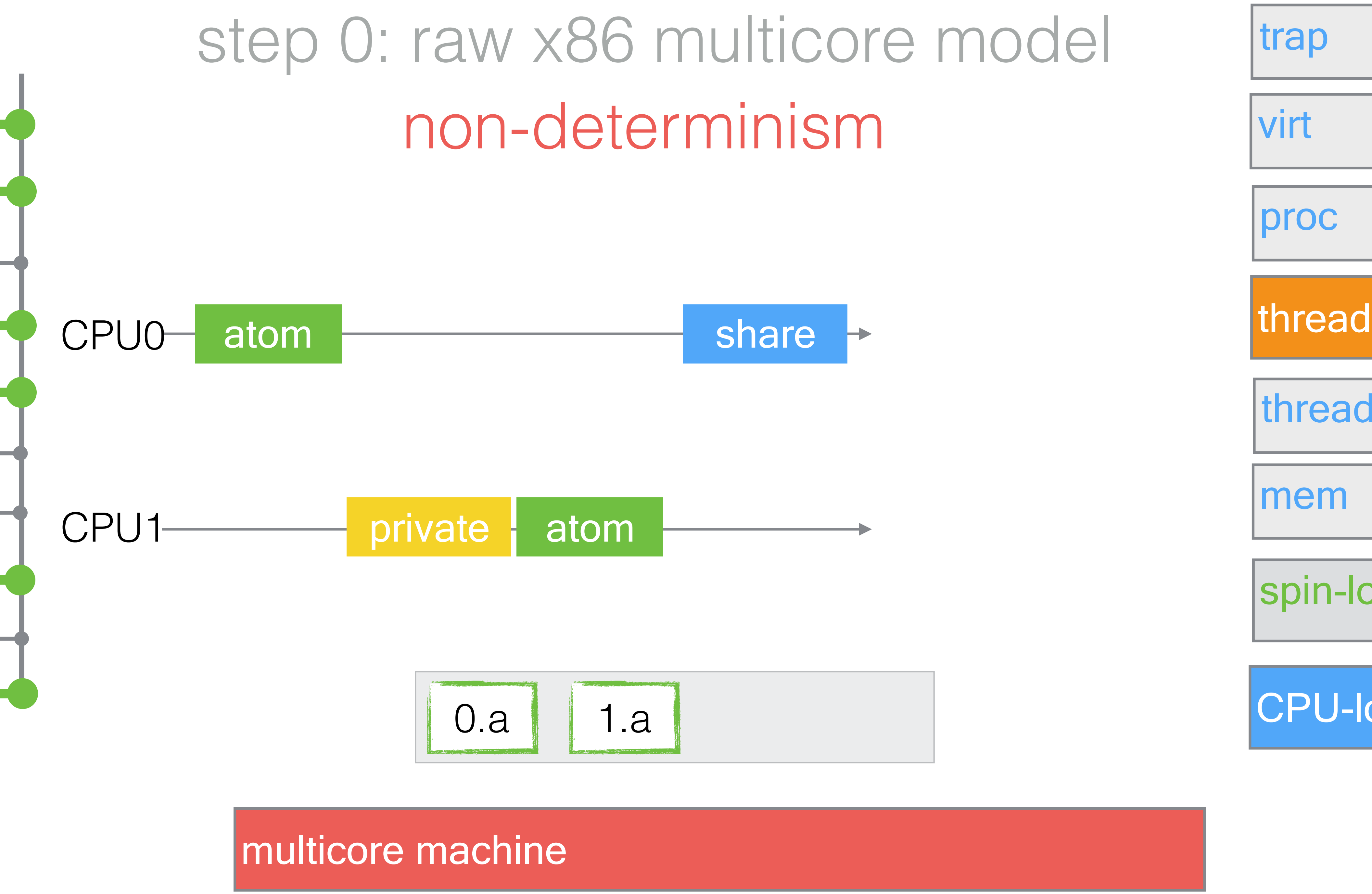


step 0: raw x86 multicore model



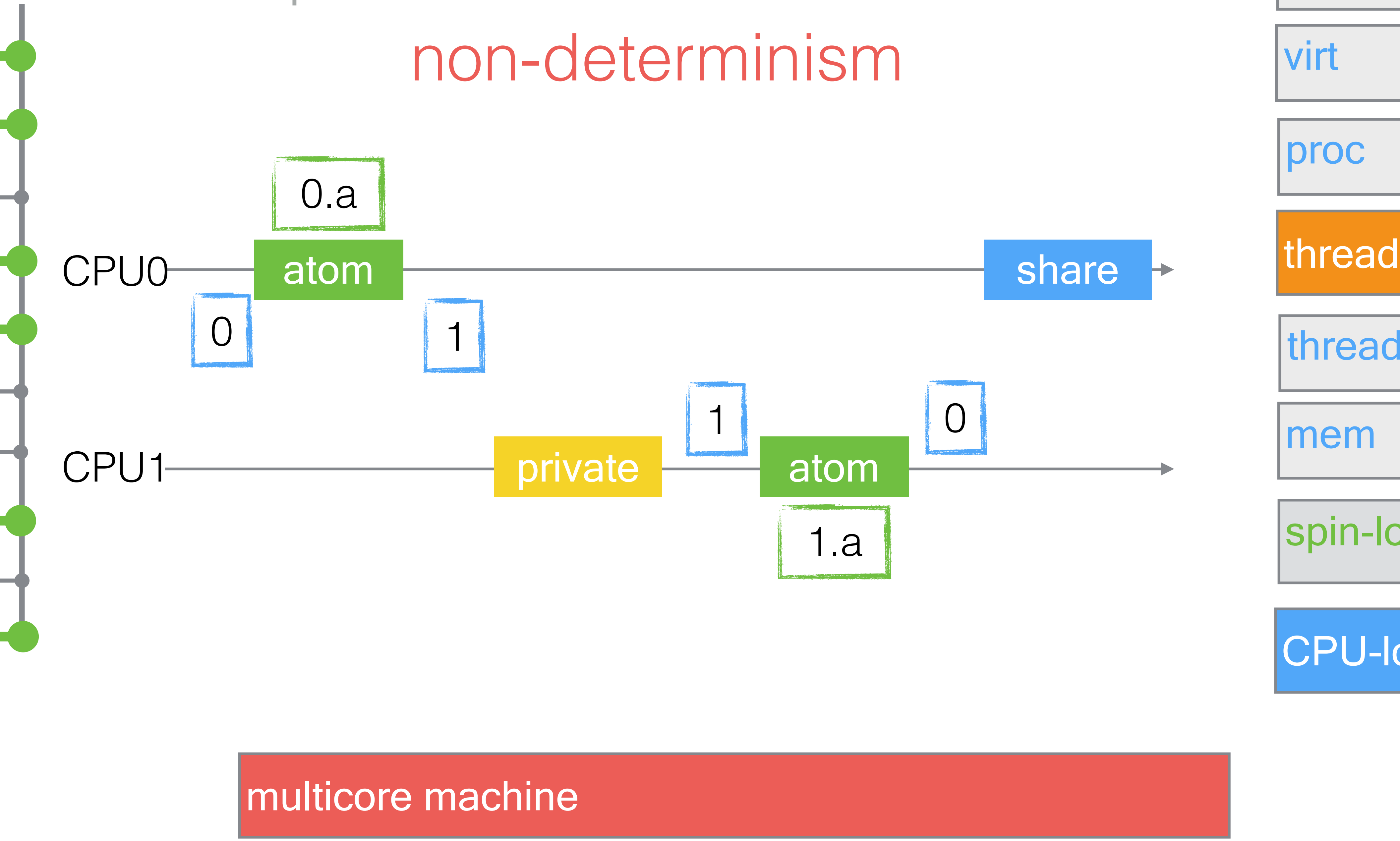
step 0: raw x86 multicore model

non-determinism



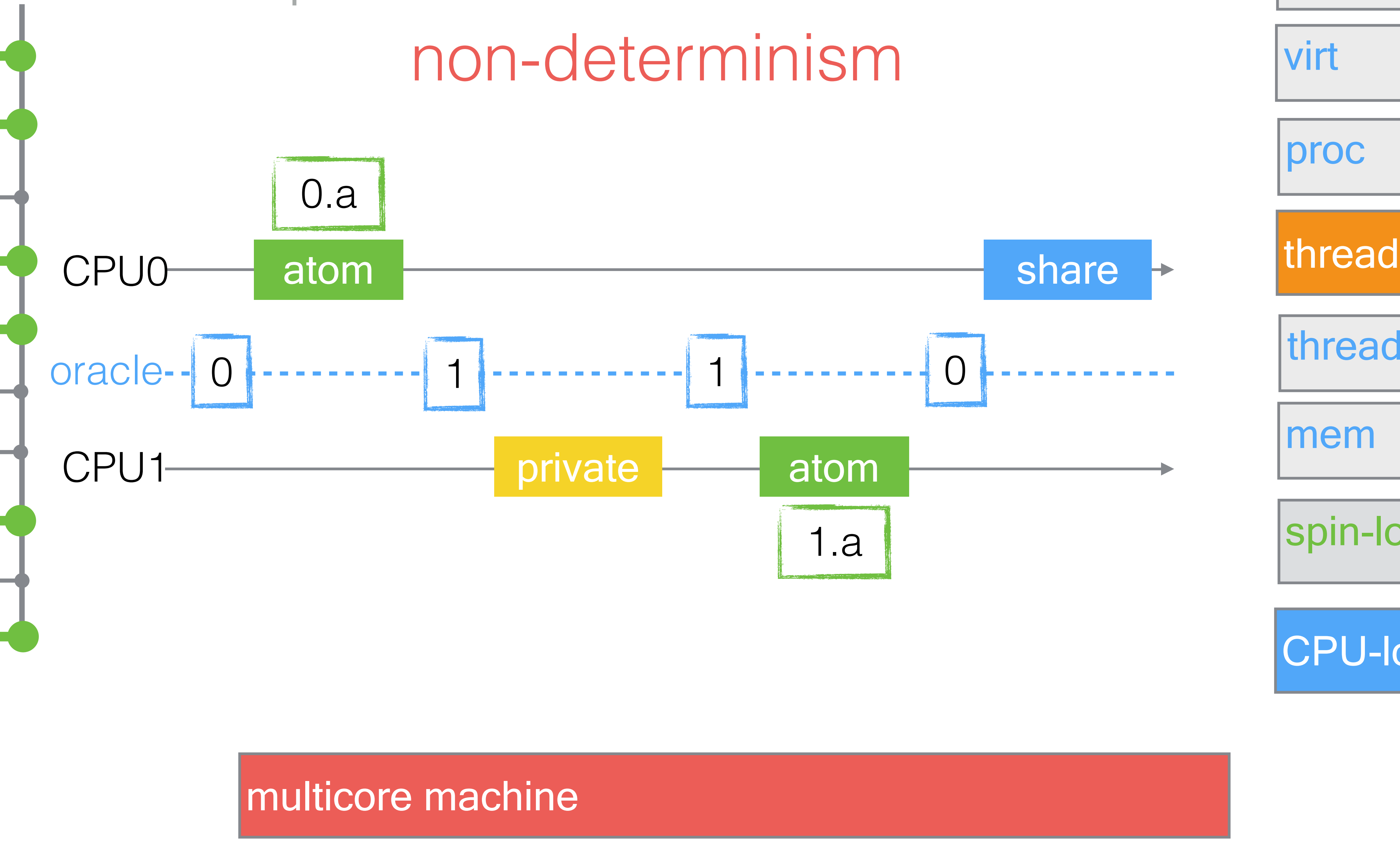
step 0: raw x86 multicore model

non-determinism



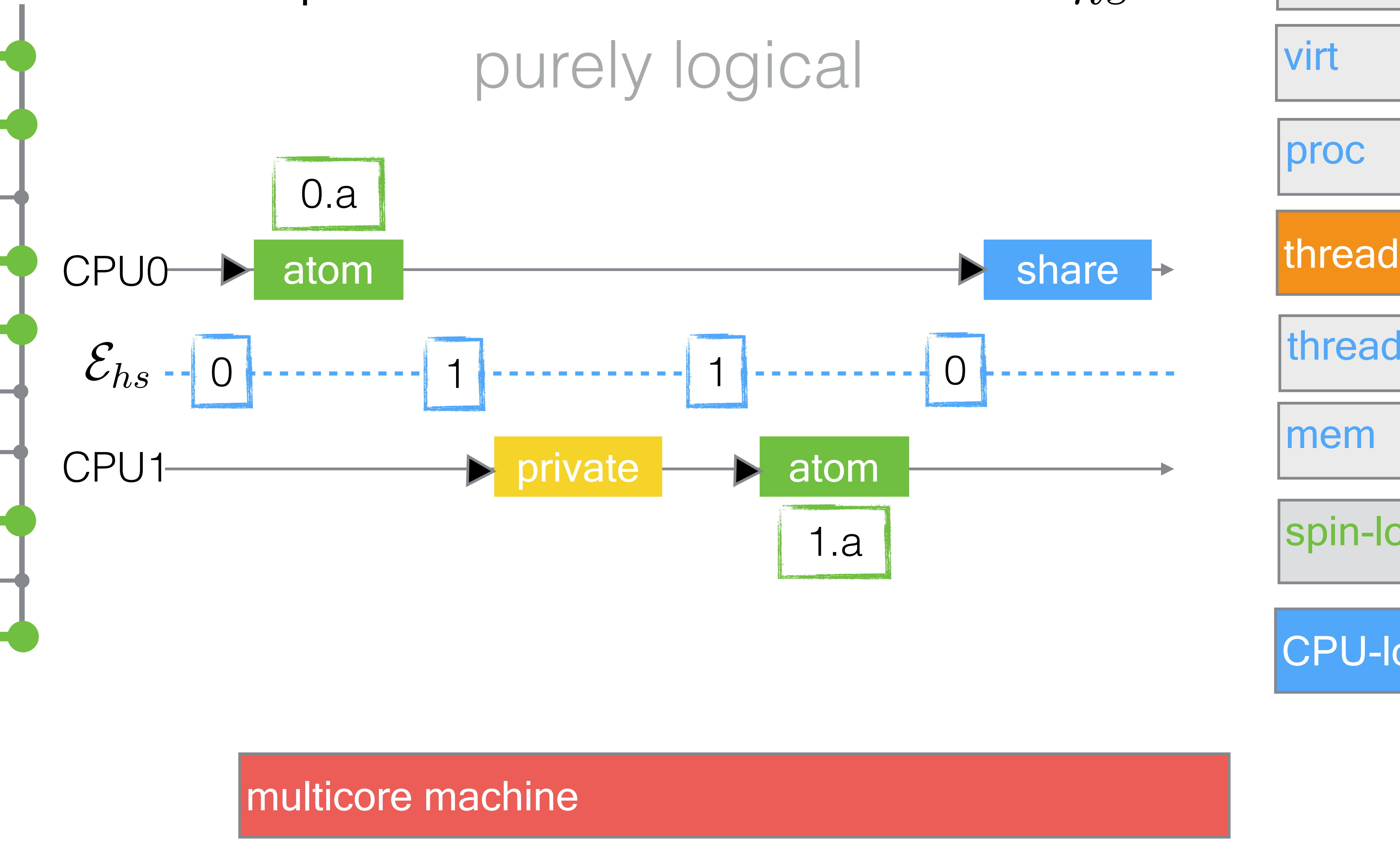
step 0: raw x86 multicore model

non-determinism



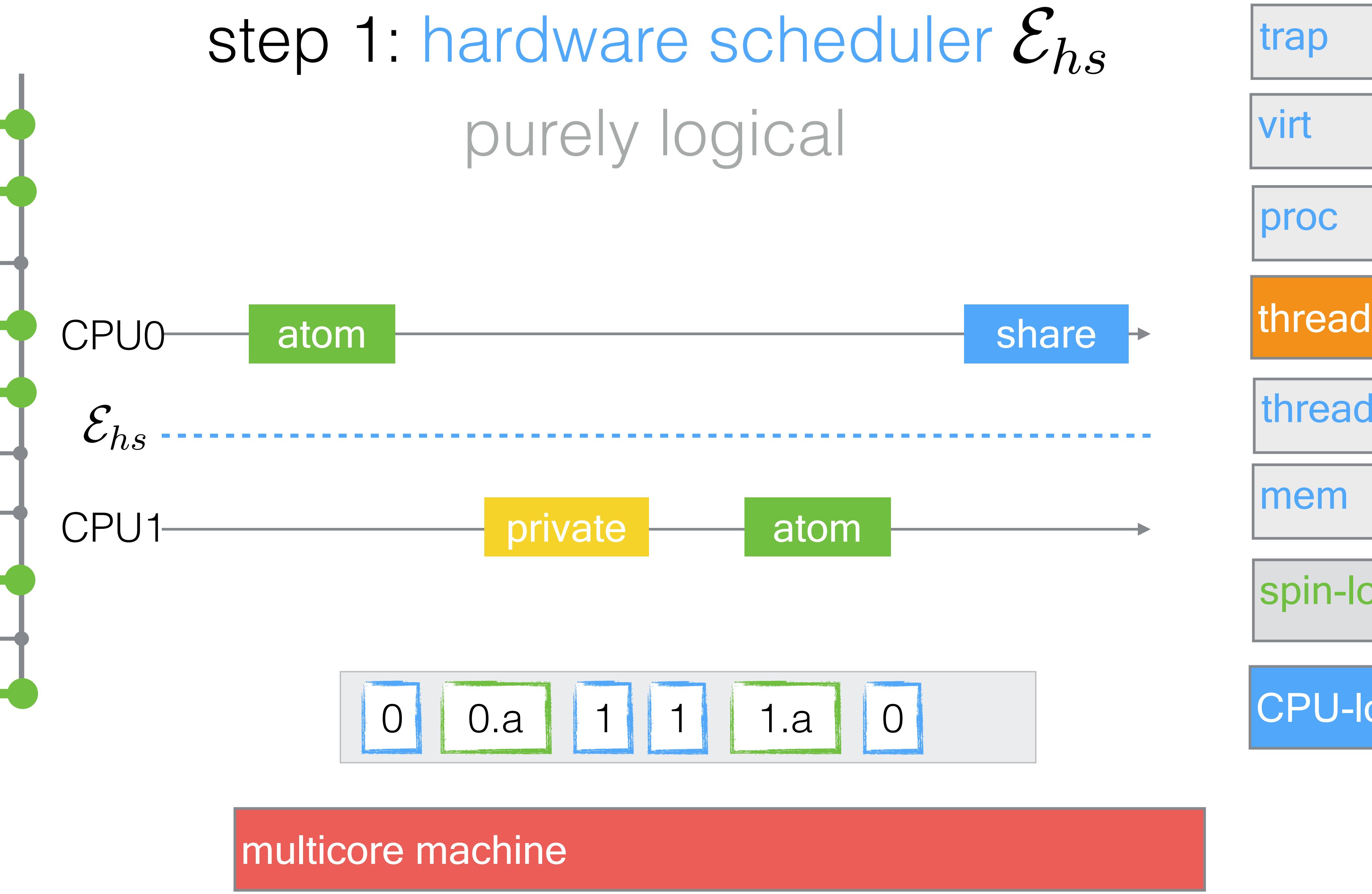
step 1: hardware scheduler $\mathcal{E}_{hs}$

purely logical



step 1: hardware scheduler $\mathcal{E}_{hs}$

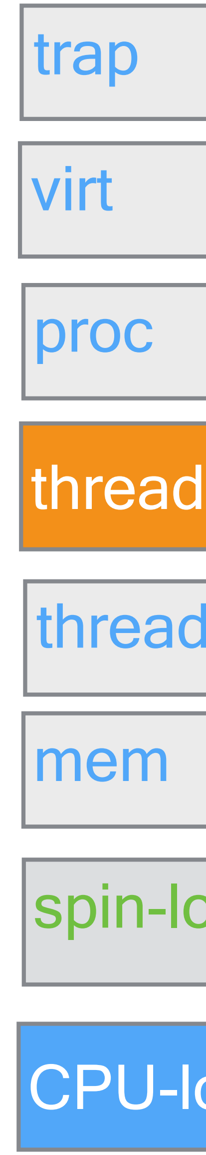
purely logical



step 1: hardware scheduler

purely logical

$\mathcal{E}_{hs}$



step 1: hardware scheduler

trap

virt

proc

thread

thread

mem

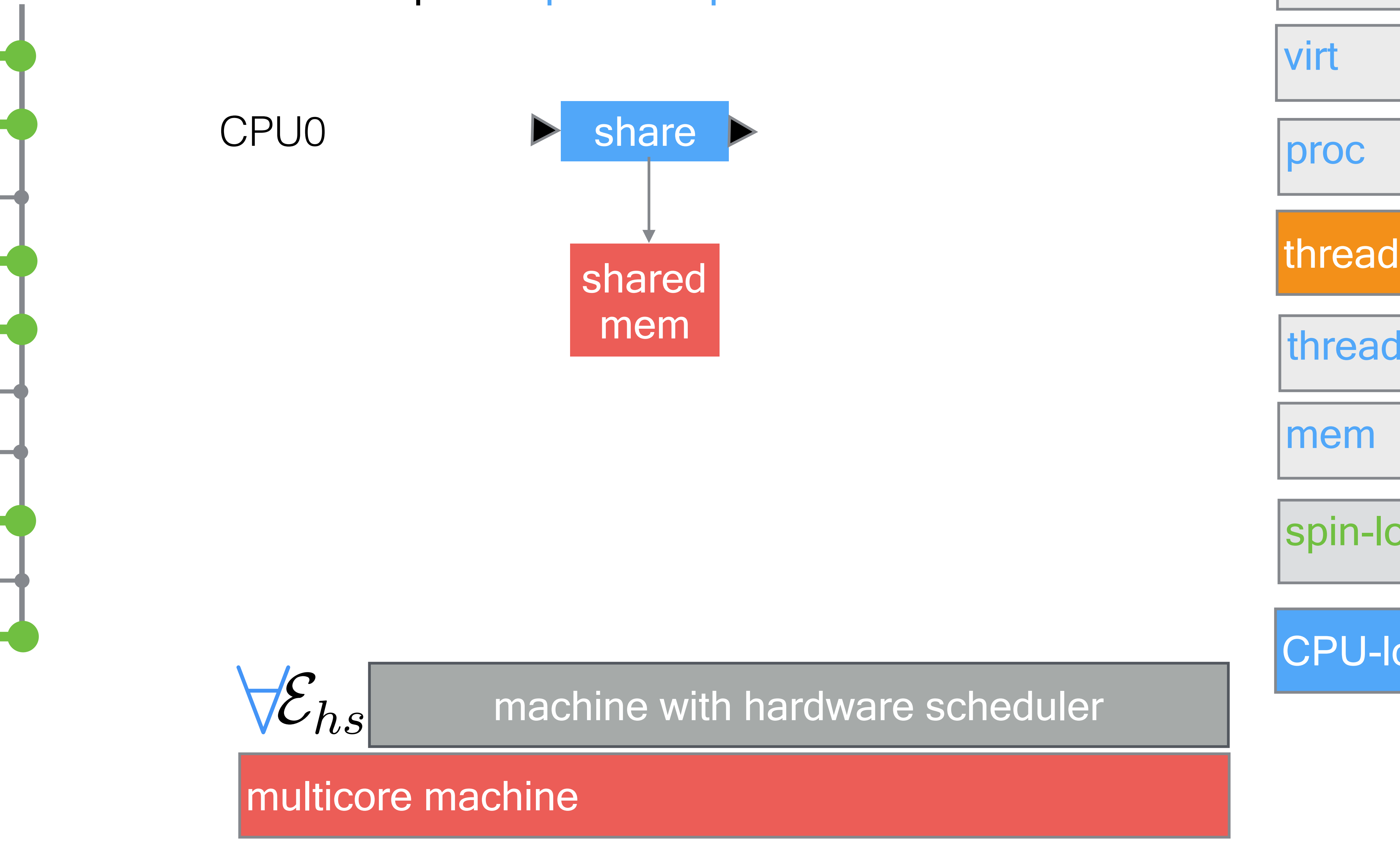
spin-lo

CPU-lo

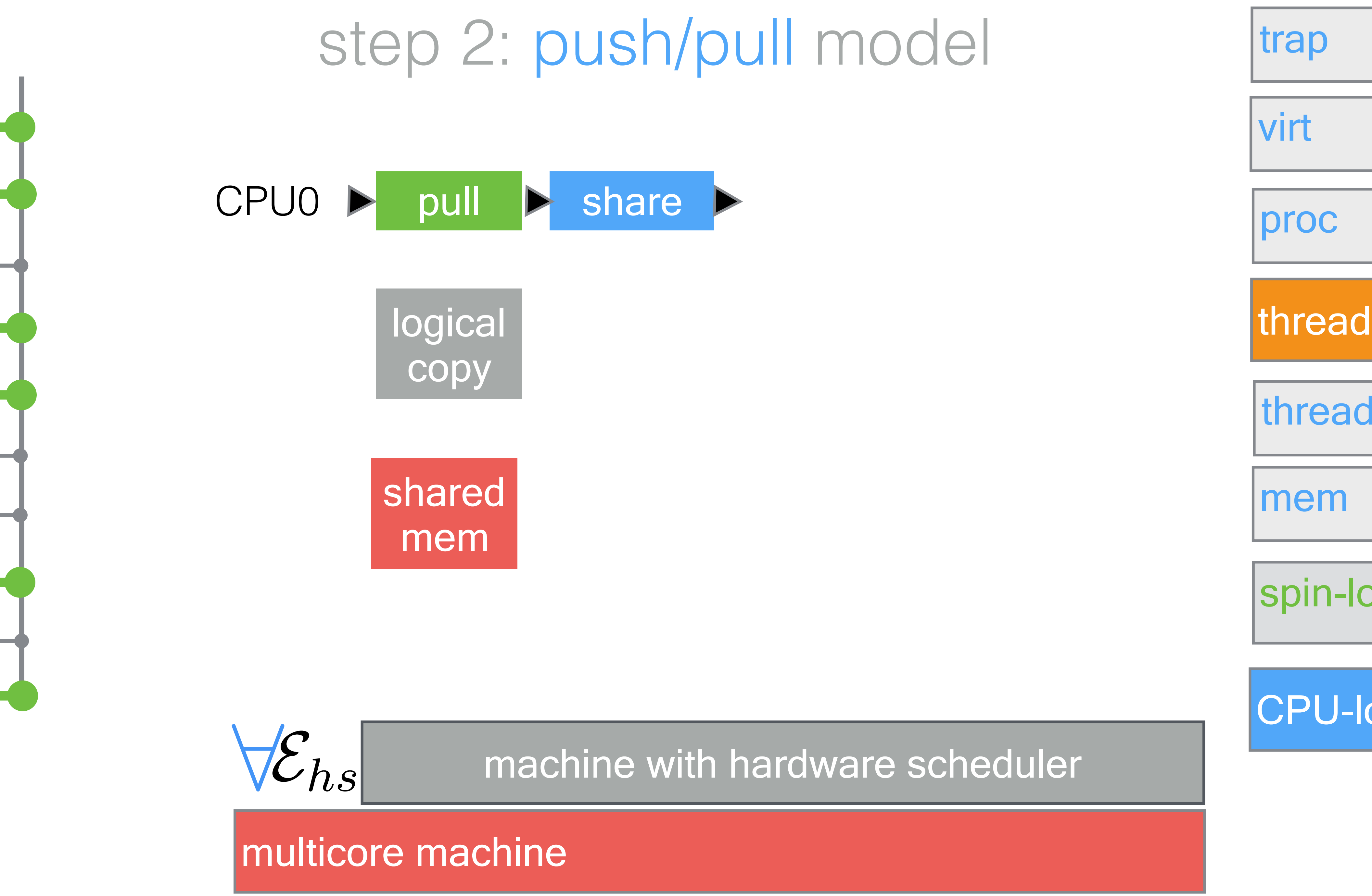
$\forall \mathcal{E}_{hs}$ machine with hardware scheduler

multicore machine

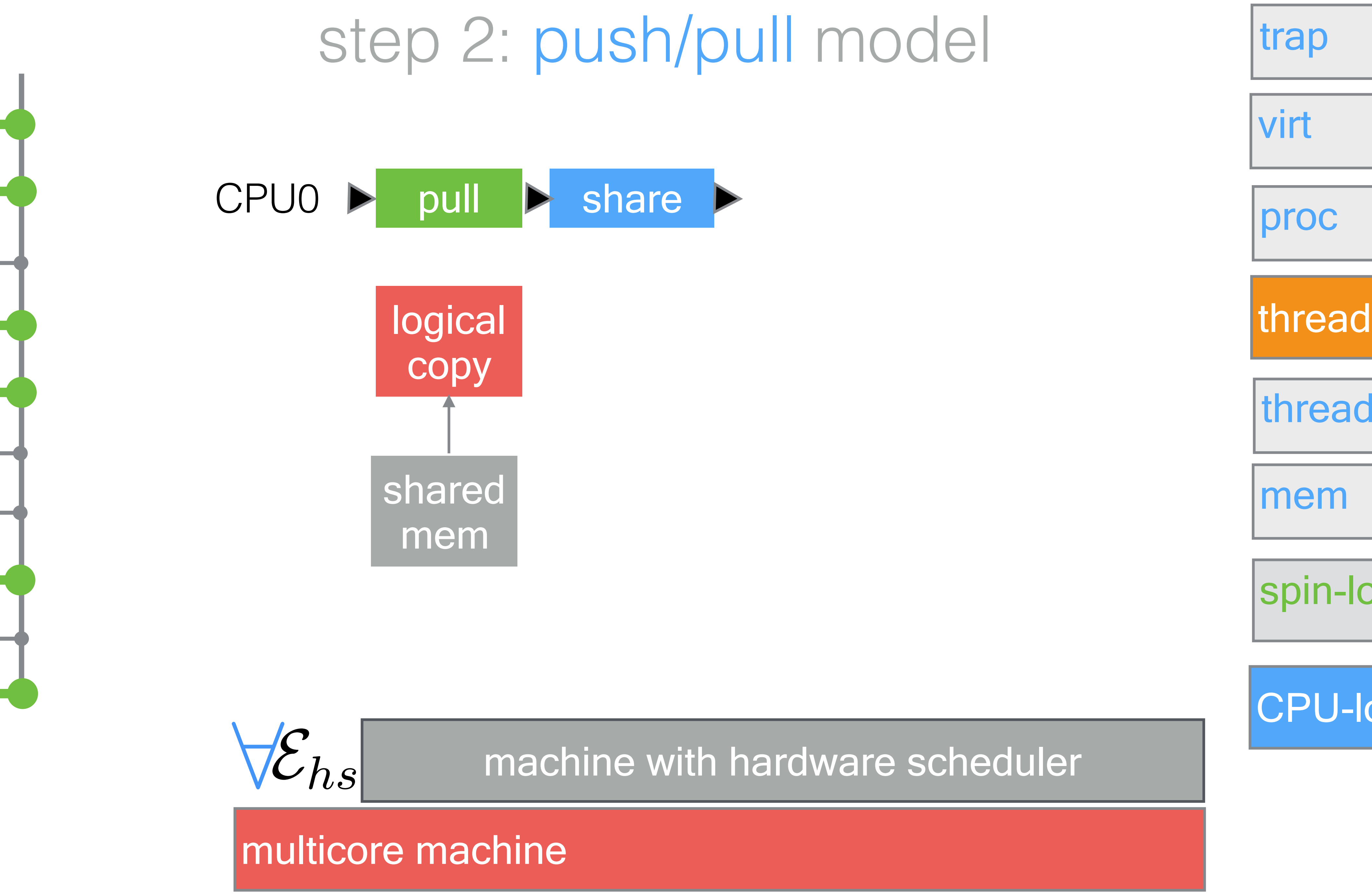
step 2: push/pull model



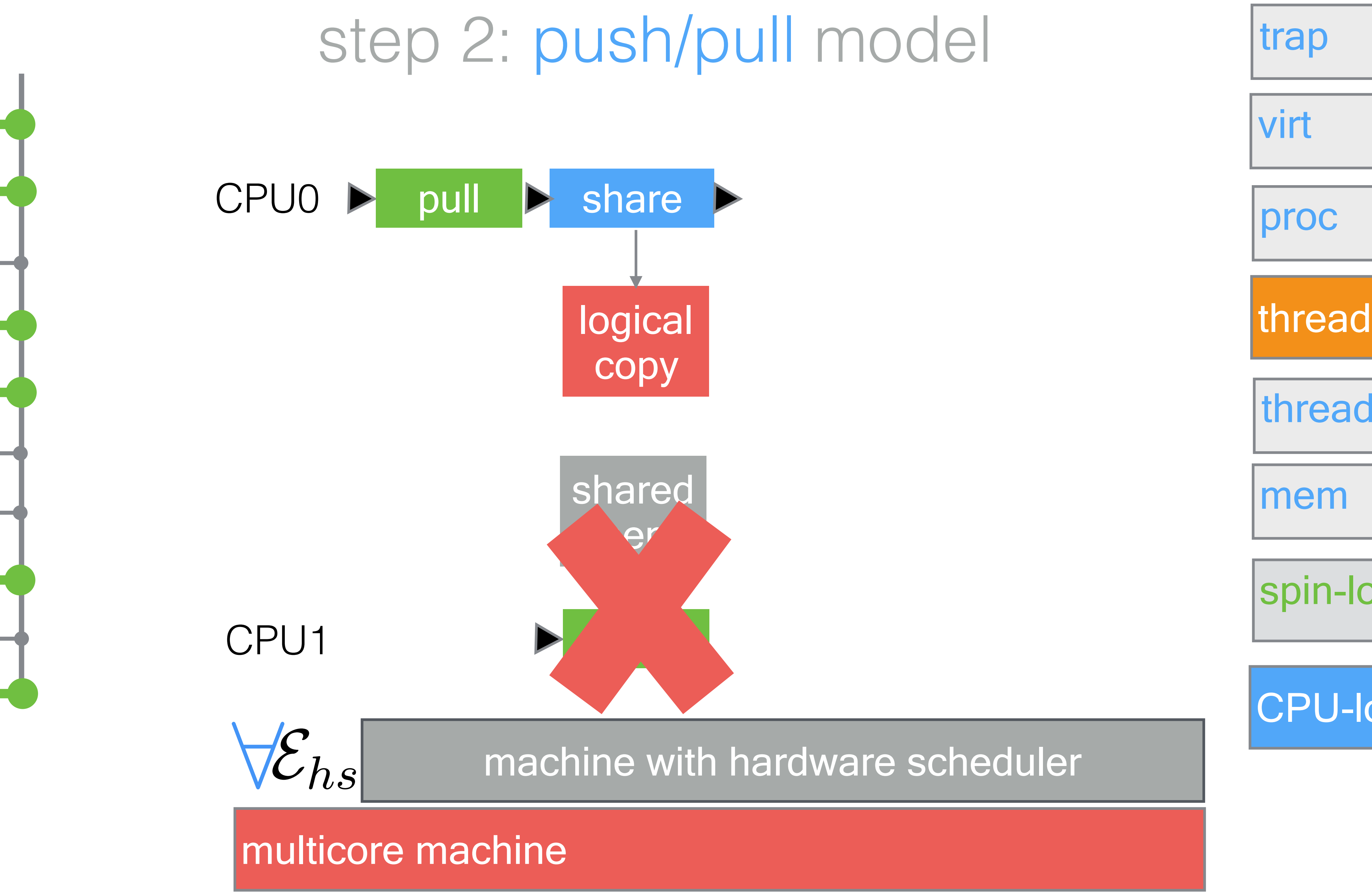
step 2: push/pull model



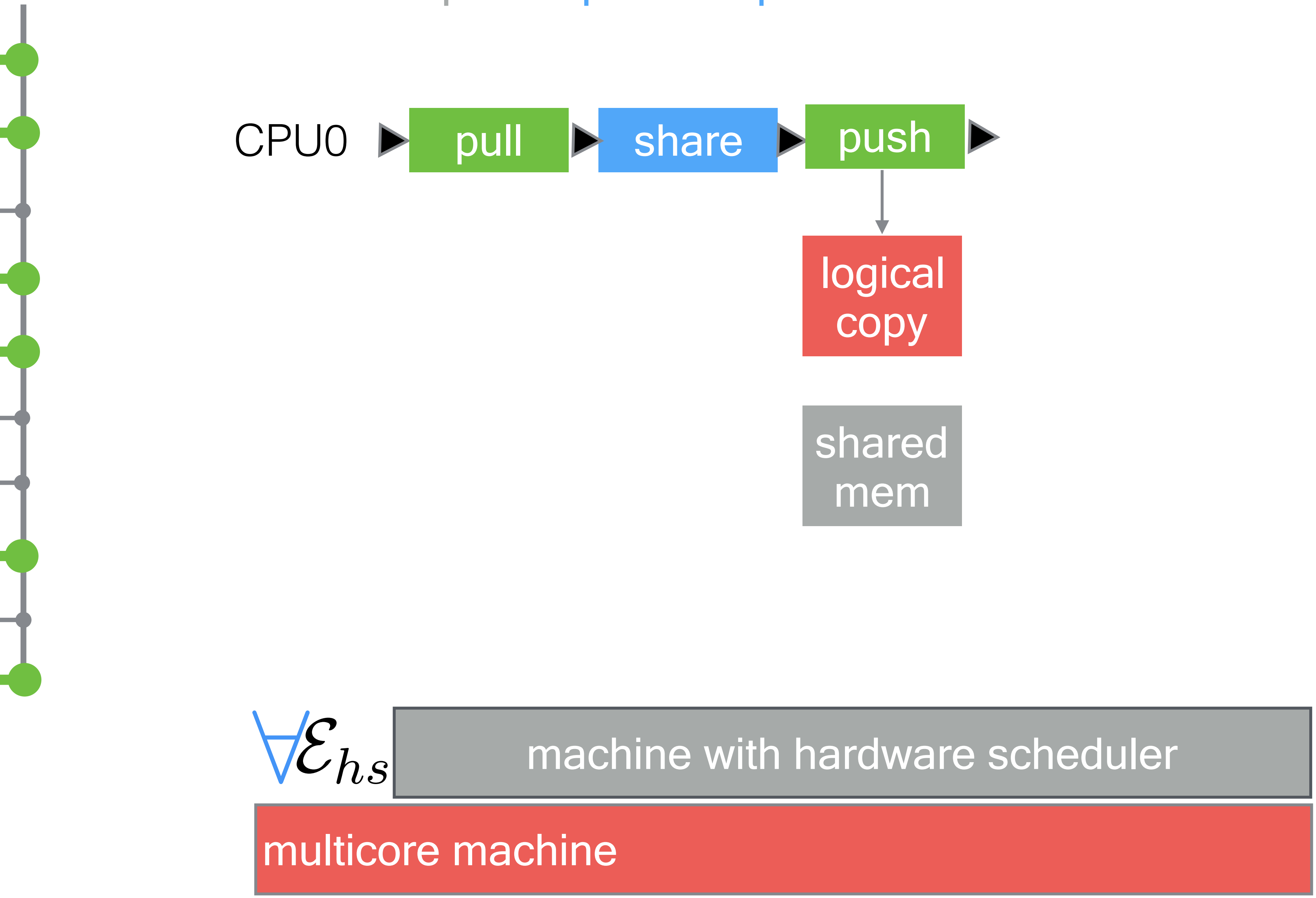
step 2: push/pull model

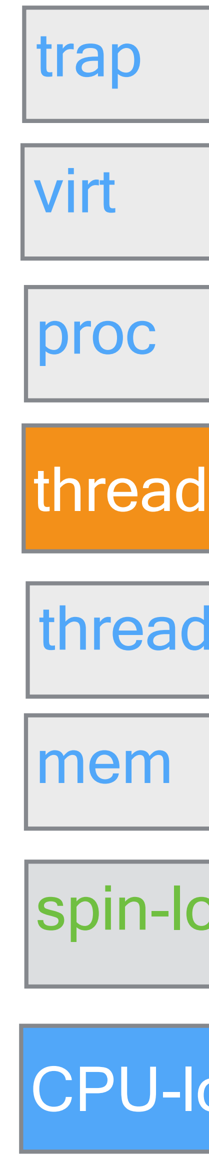


step 2: push/pull model

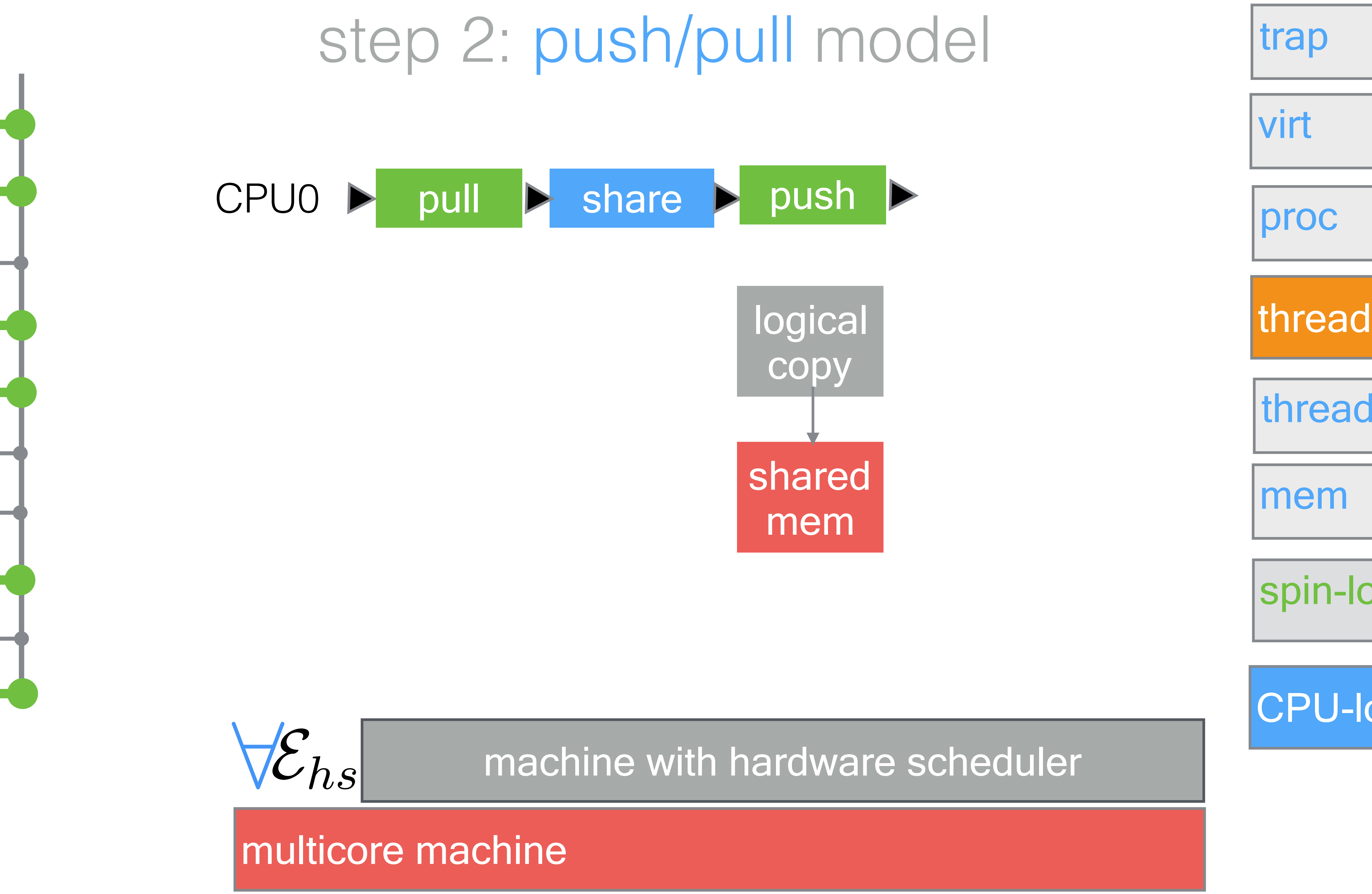


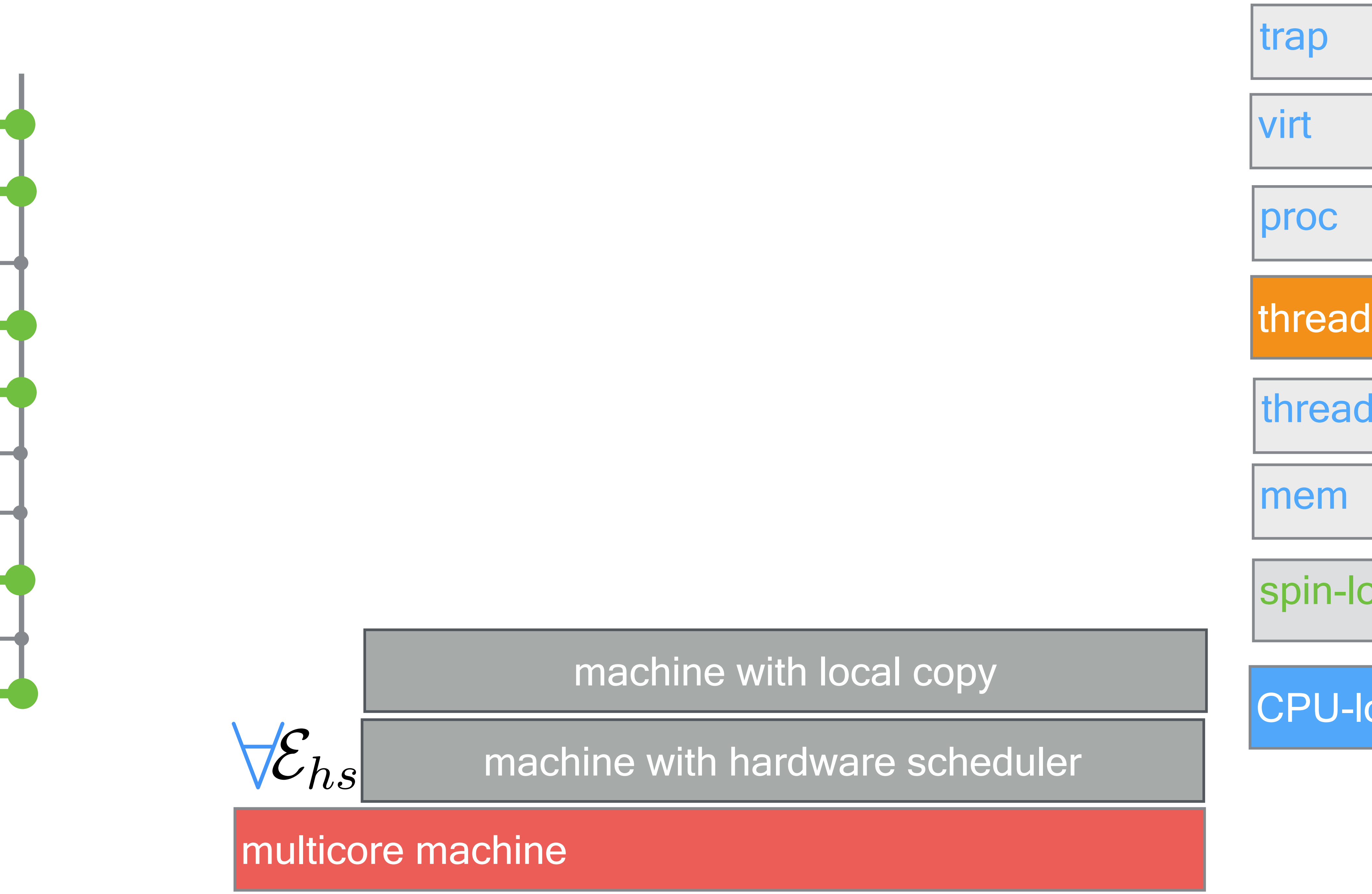
step 2: push/pull model



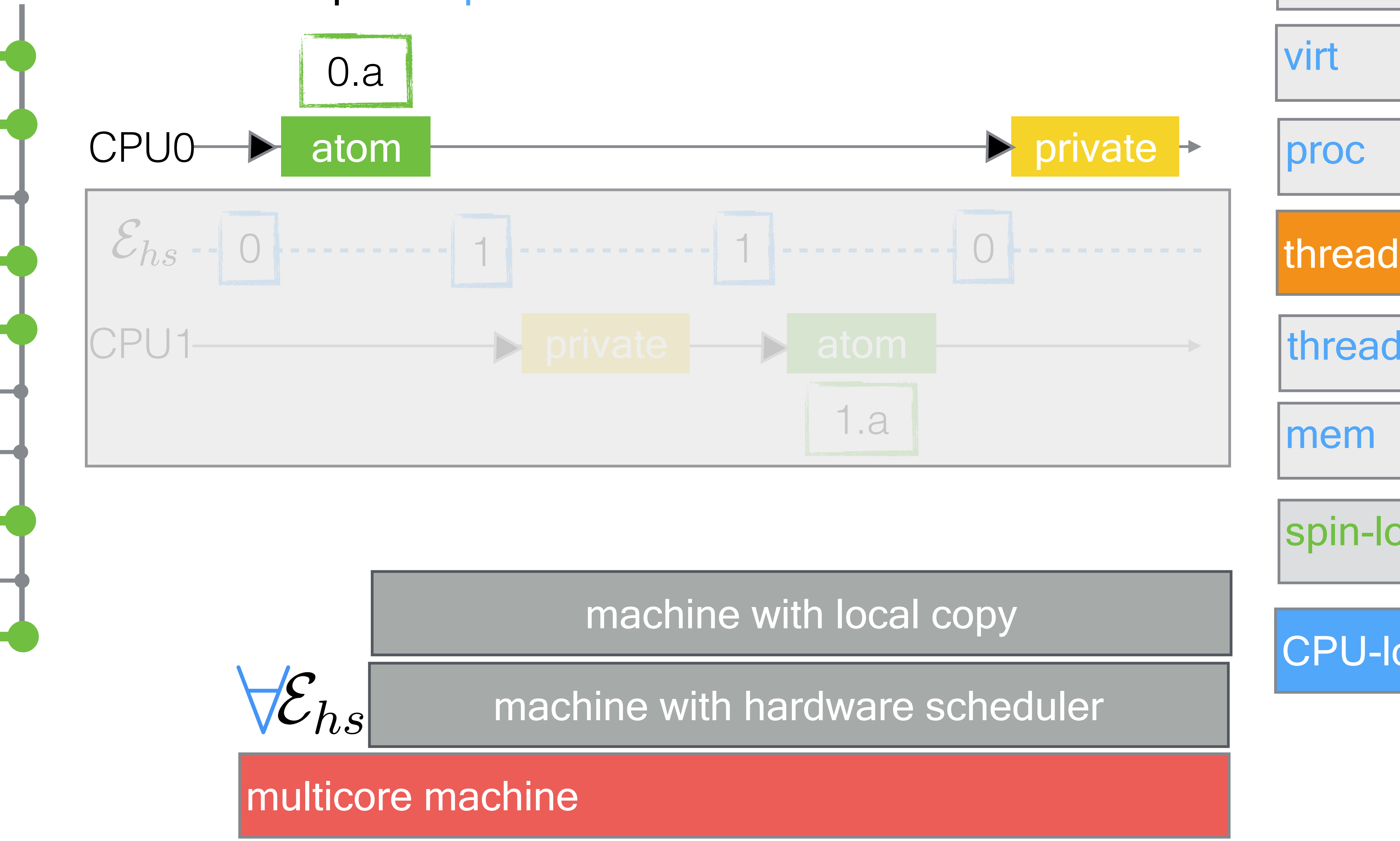


step 2: push/pull model

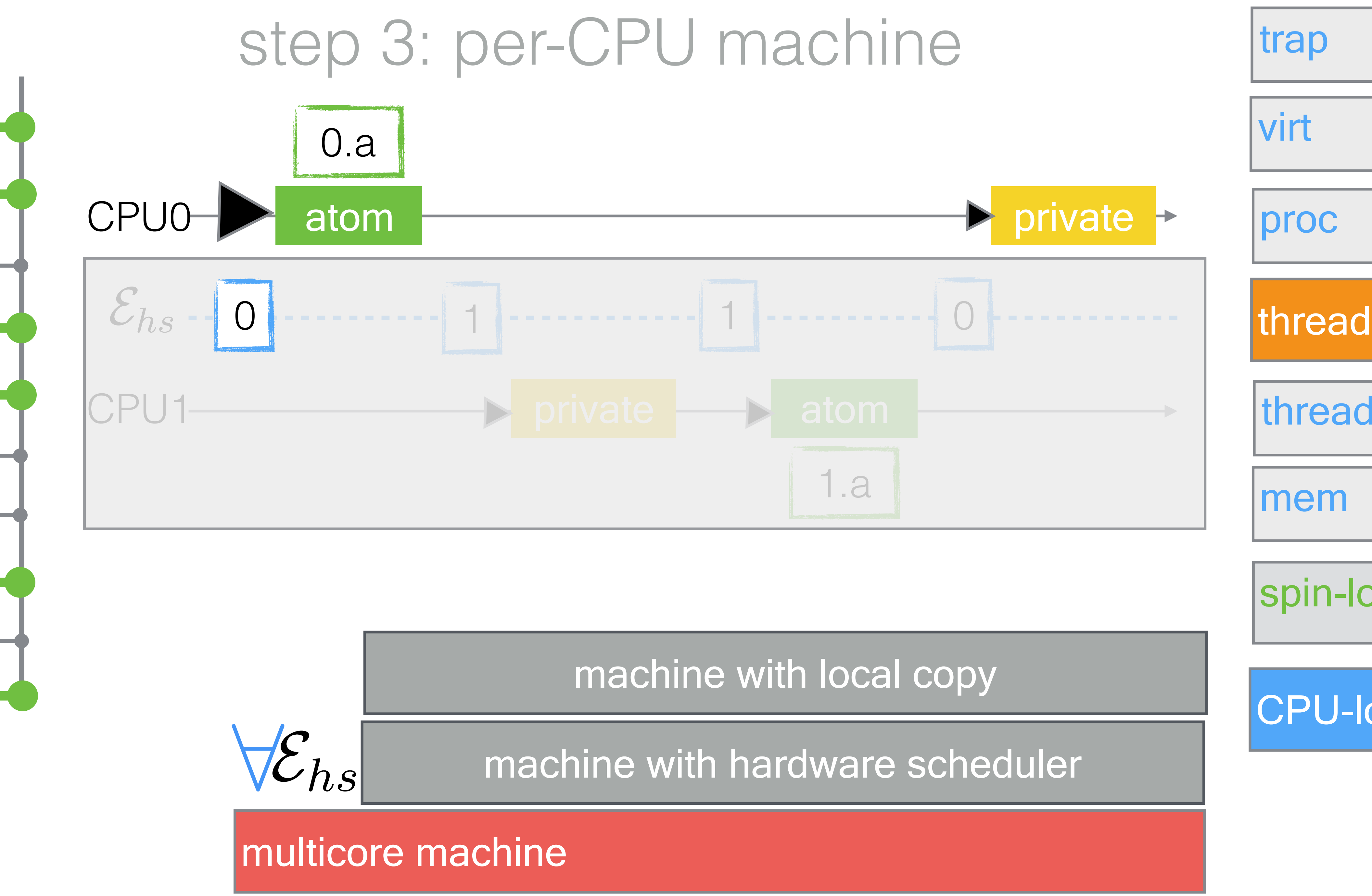



trap
virt
proc
thread
thread
mem
spin-lo
CPU-lo
machine with local copy
$\forall \mathcal{E}_{hs}$
machine with hardware scheduler
multicore machine

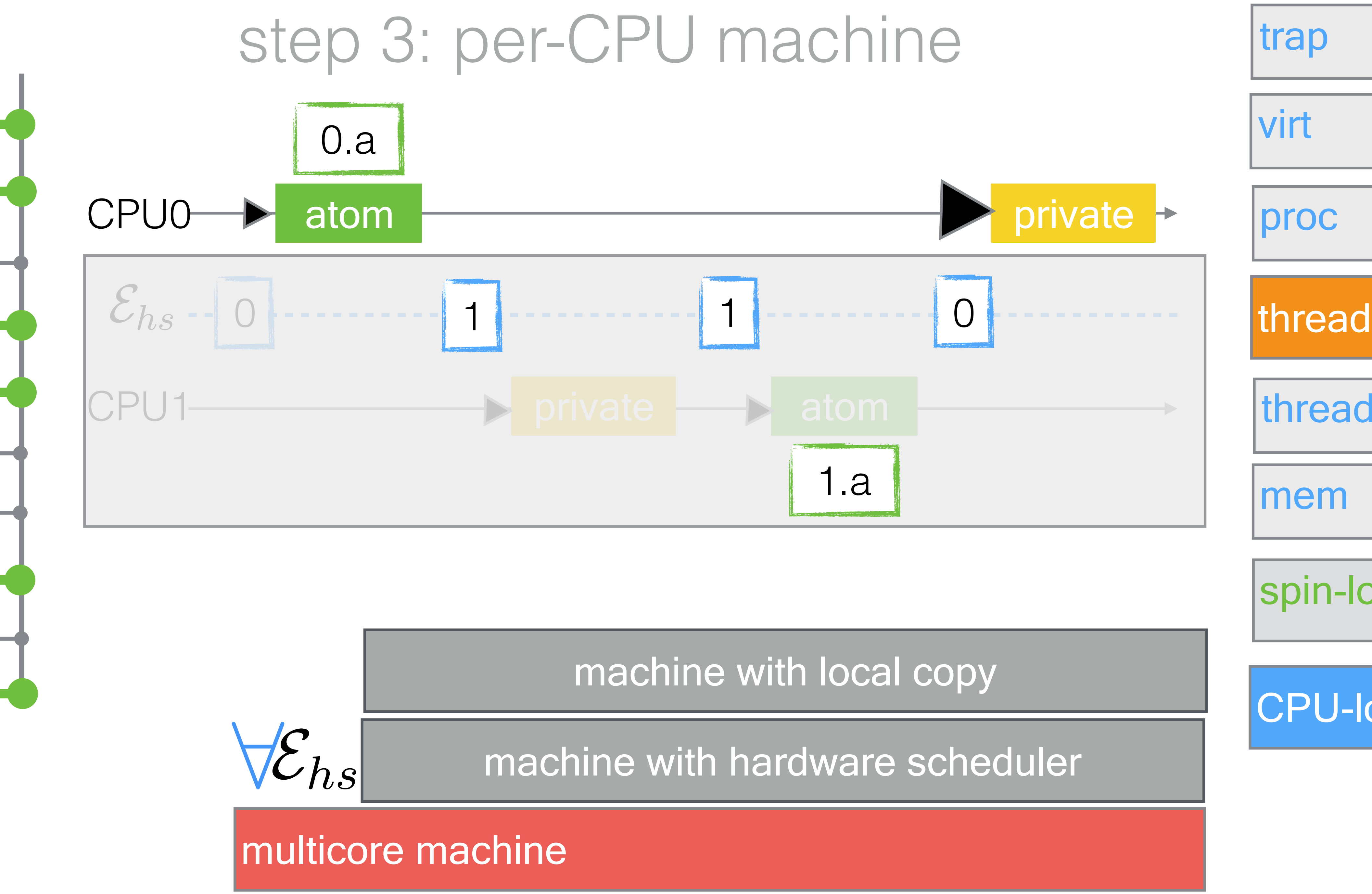
step 3: per-CPU machine



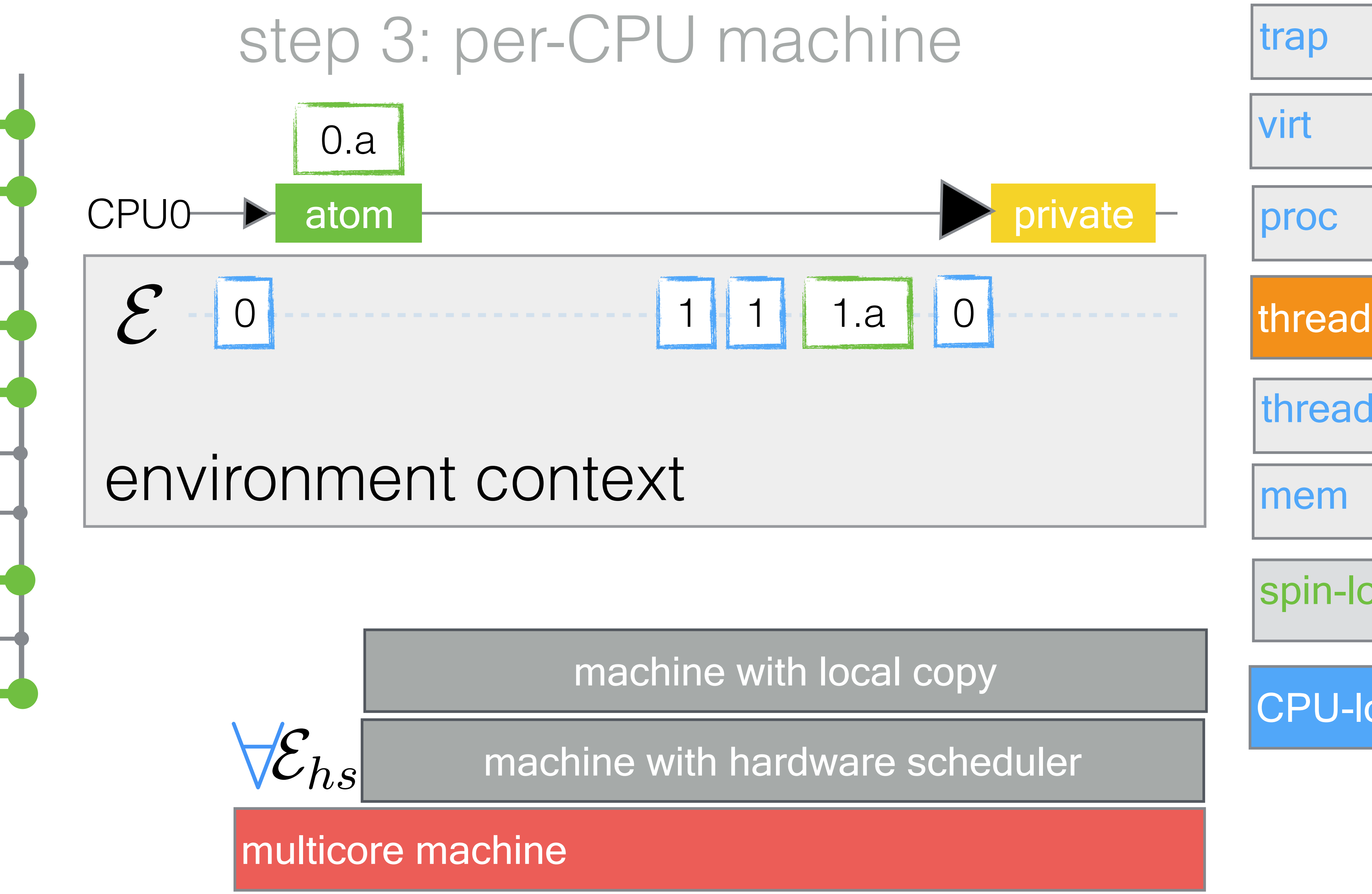
step 3: per-CPU machine

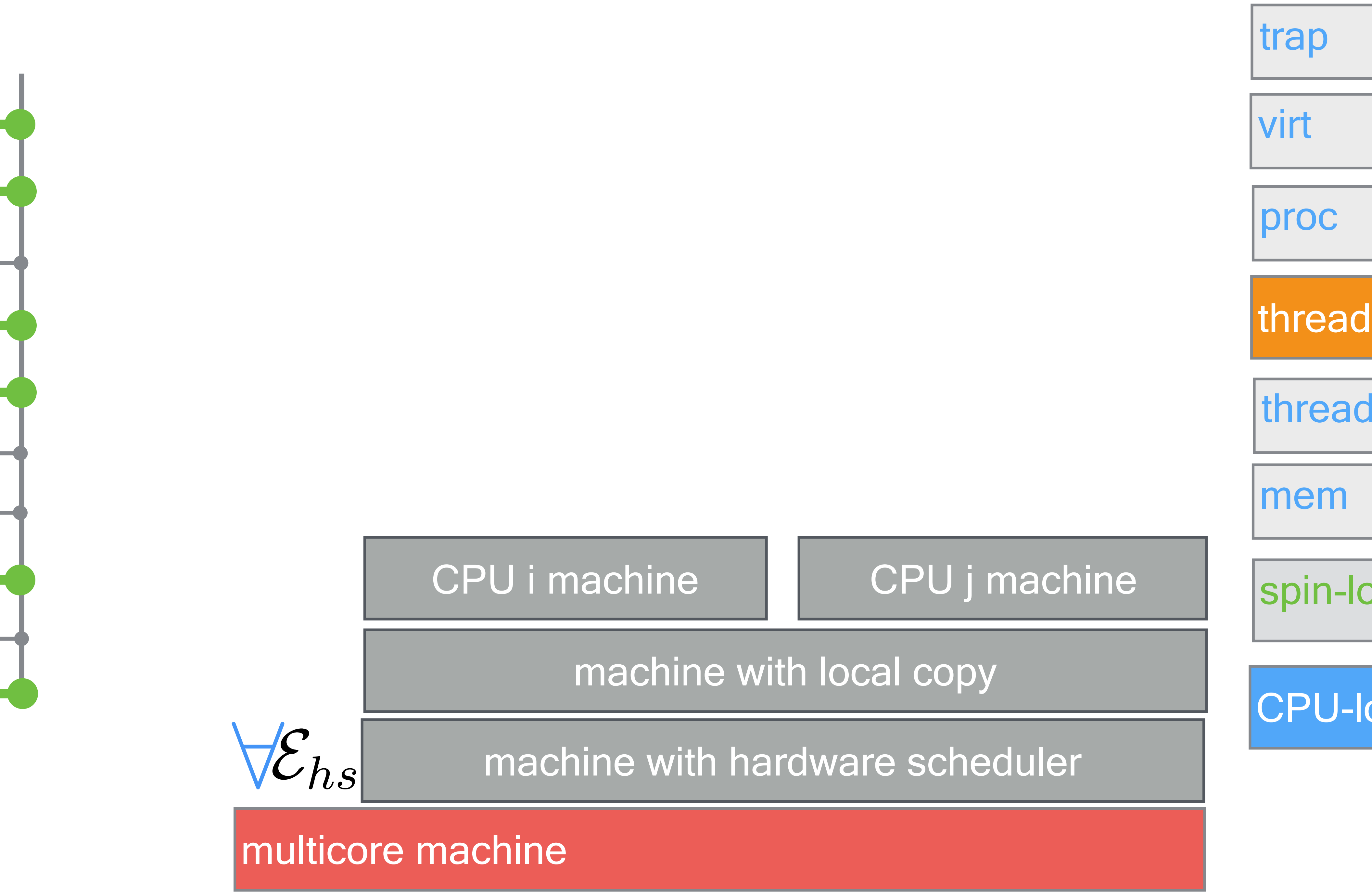


step 3: per-CPU machine

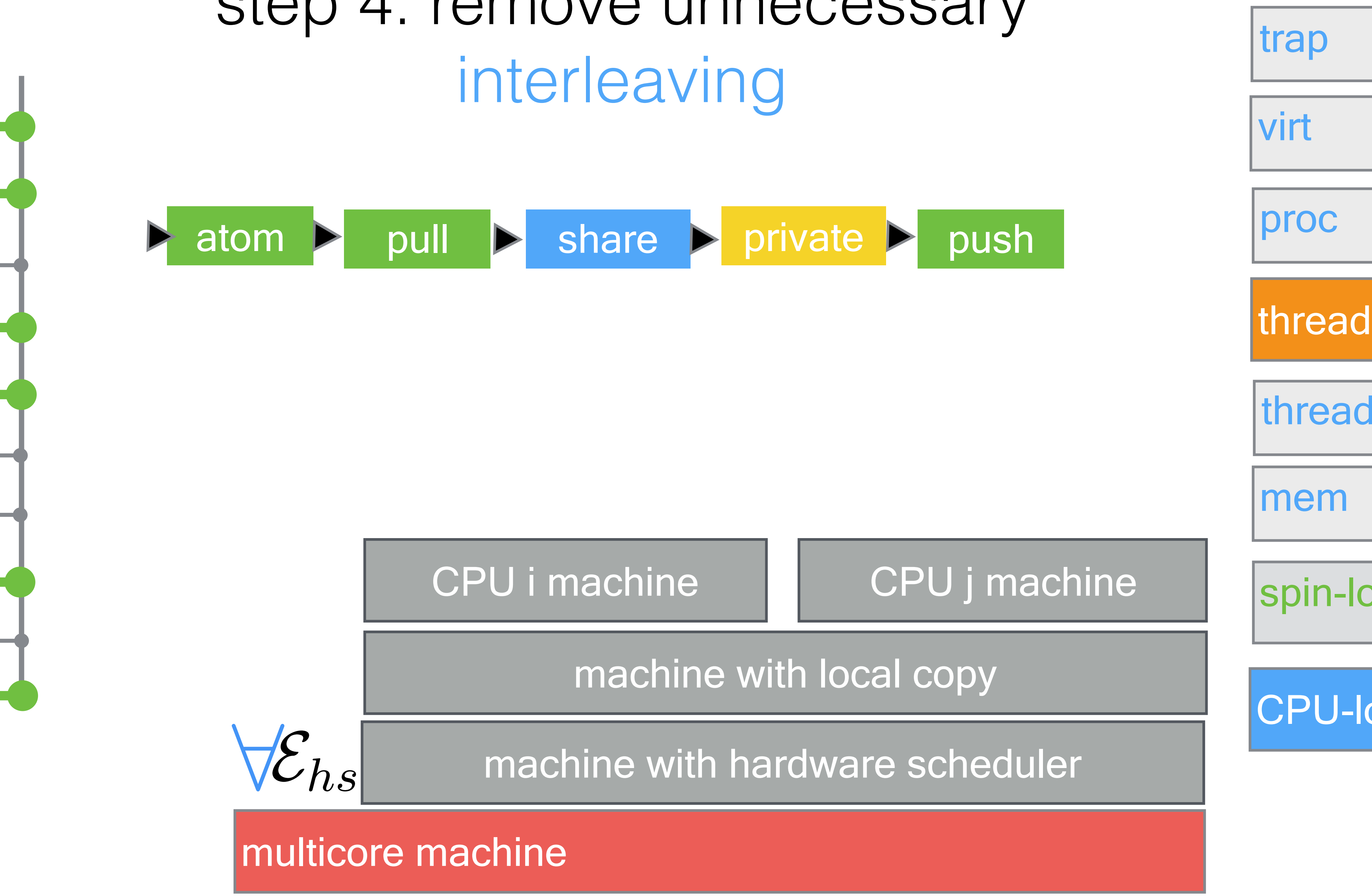


step 3: per-CPU machine

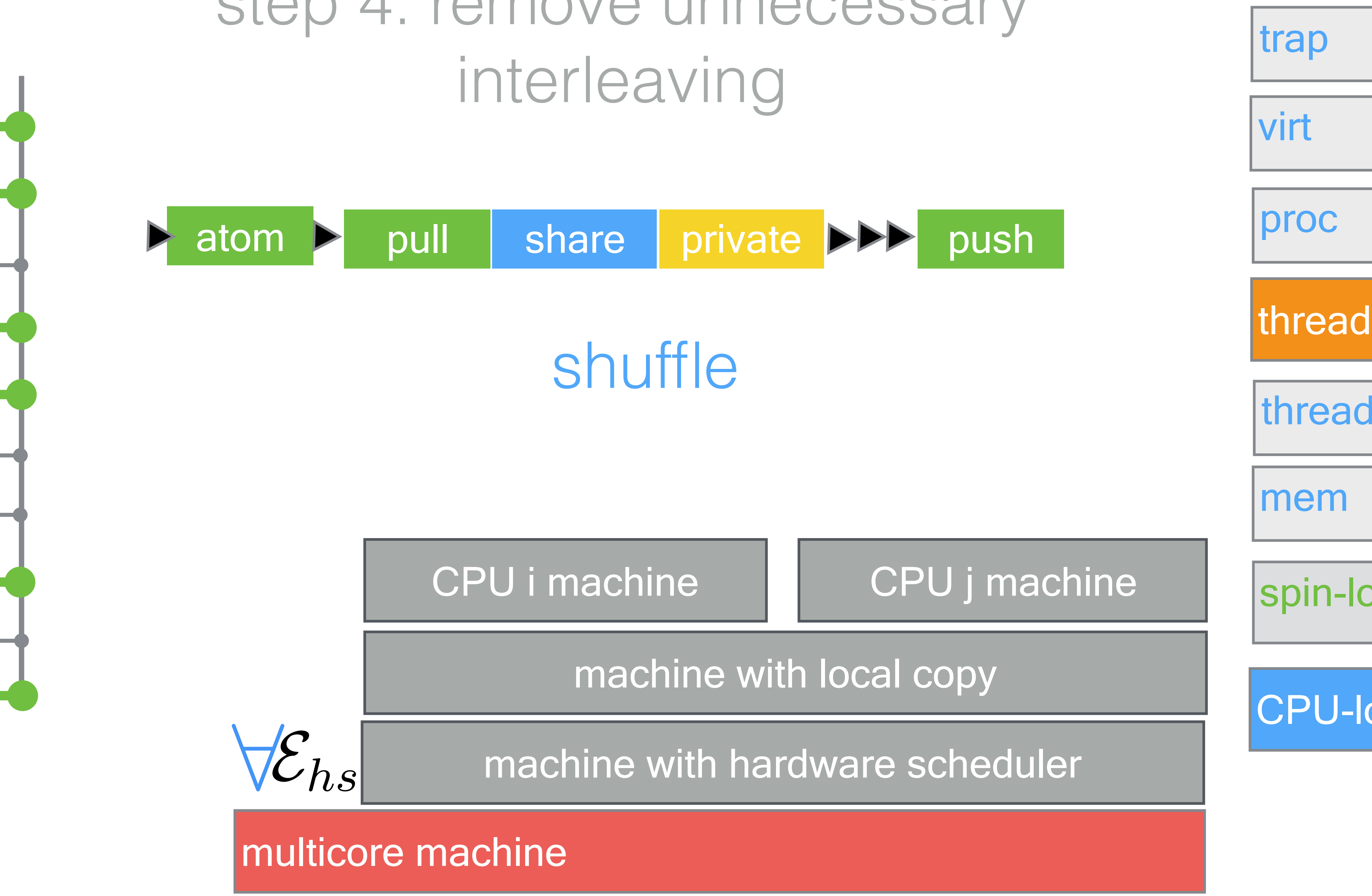



trap
virt
proc
thread
thread
mem
spin-lo
CPU-lo
CPU i machine
CPU j machine
machine with local copy
$\forall \mathcal{E}_{hs}$
machine with hardware scheduler
multicore machine

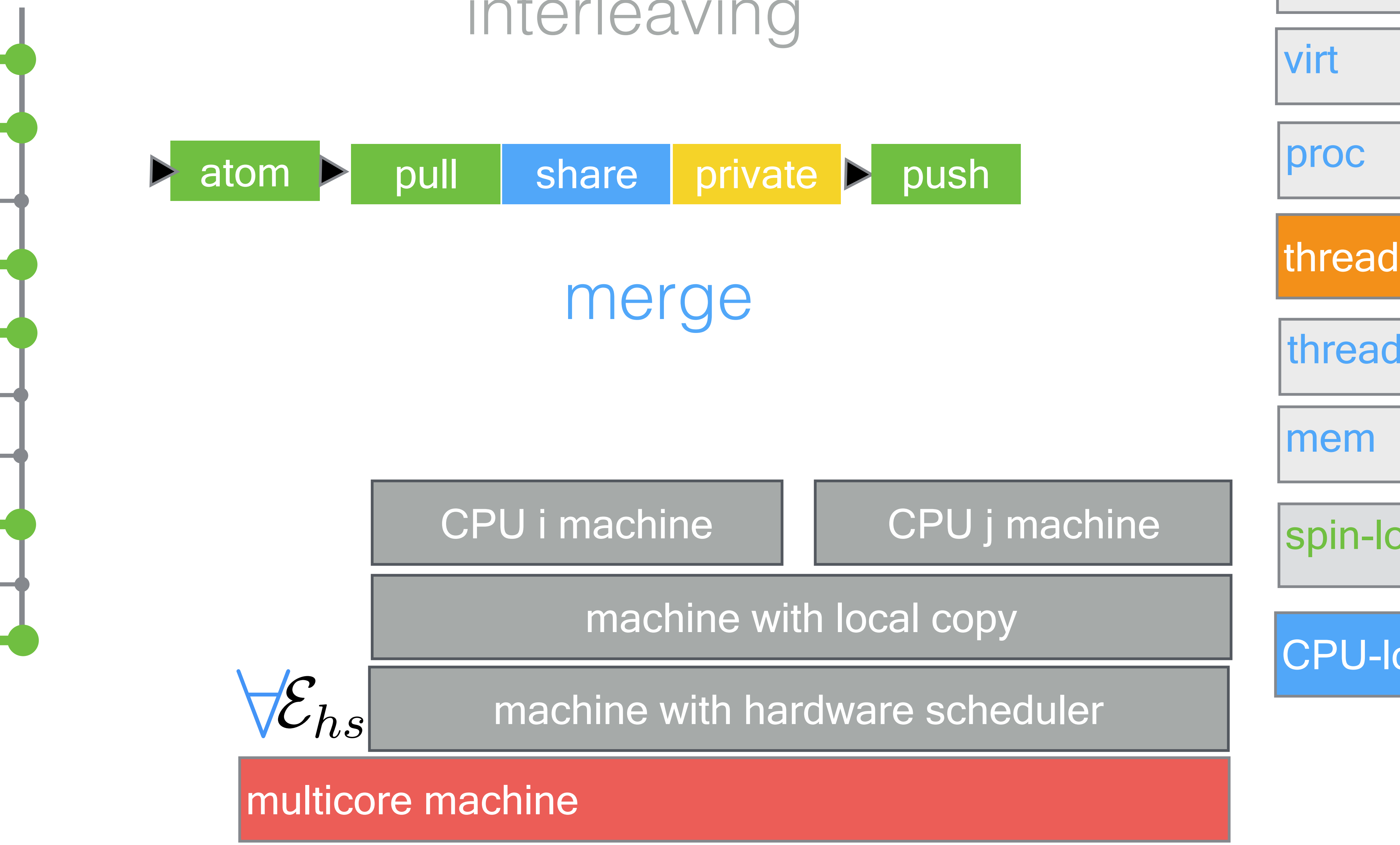
step 4: remove unnecessary interleaving

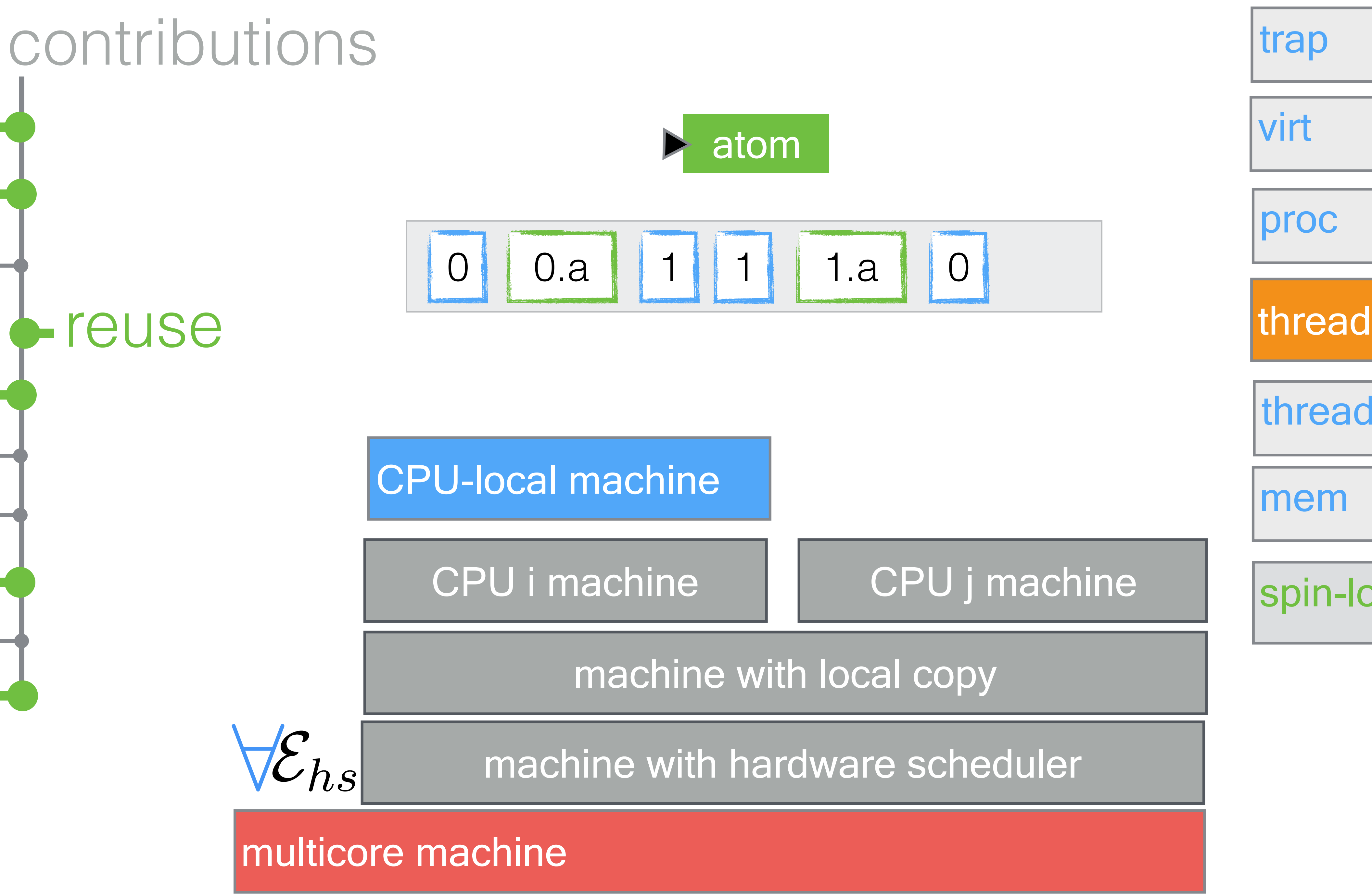


step 4: remove unnecessary interleaving

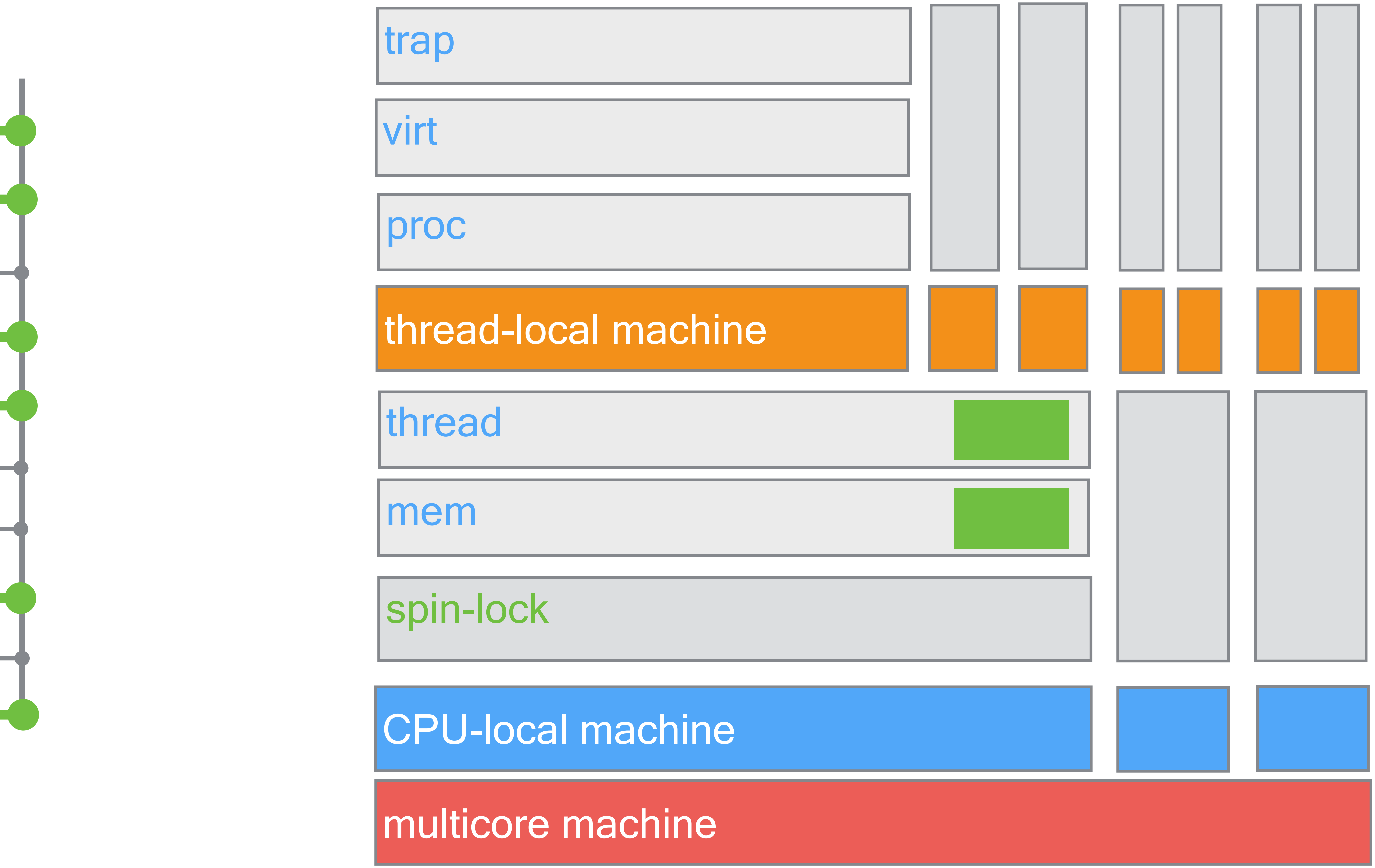


step 4: remove unnecessary interleaving

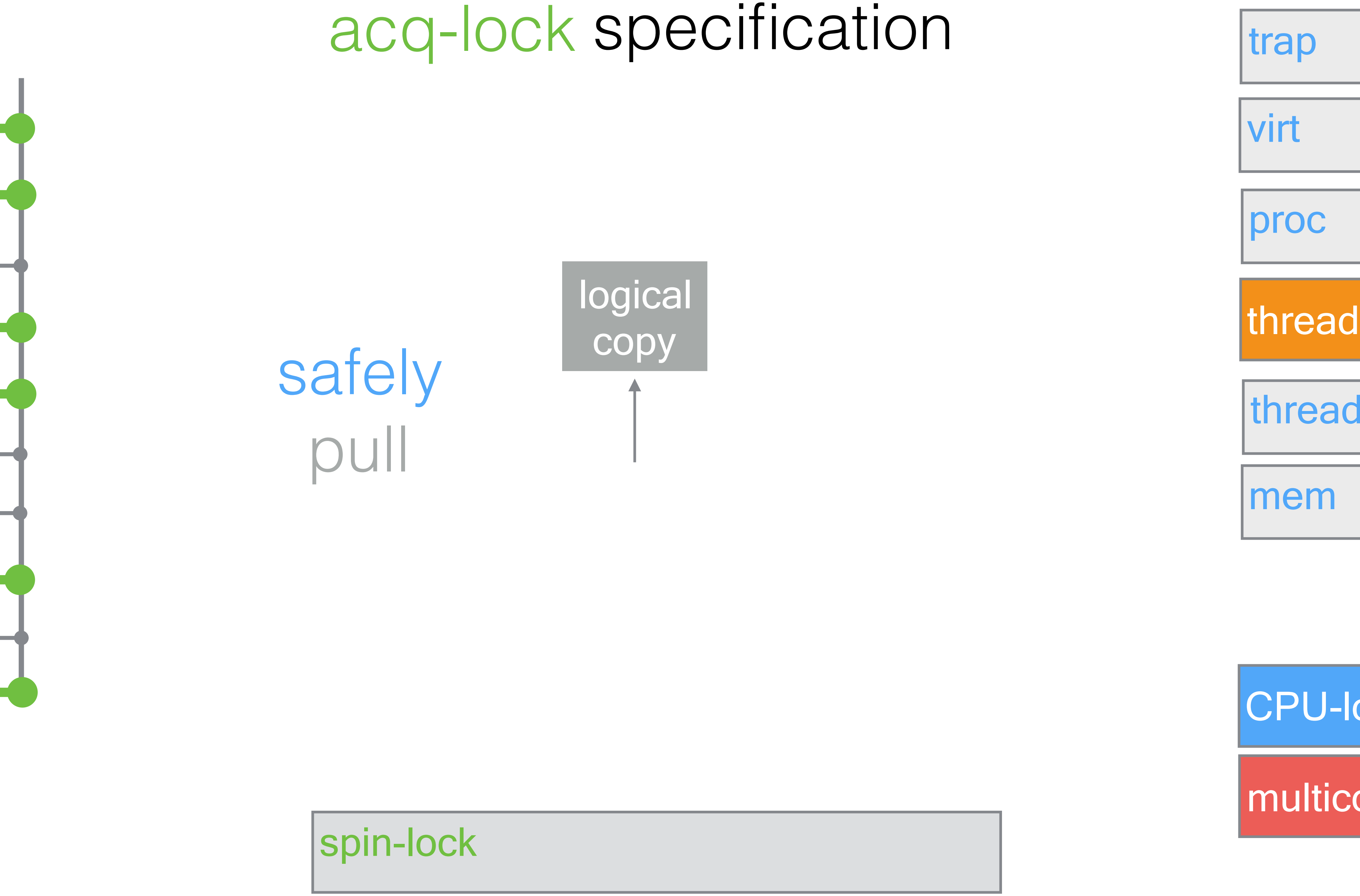


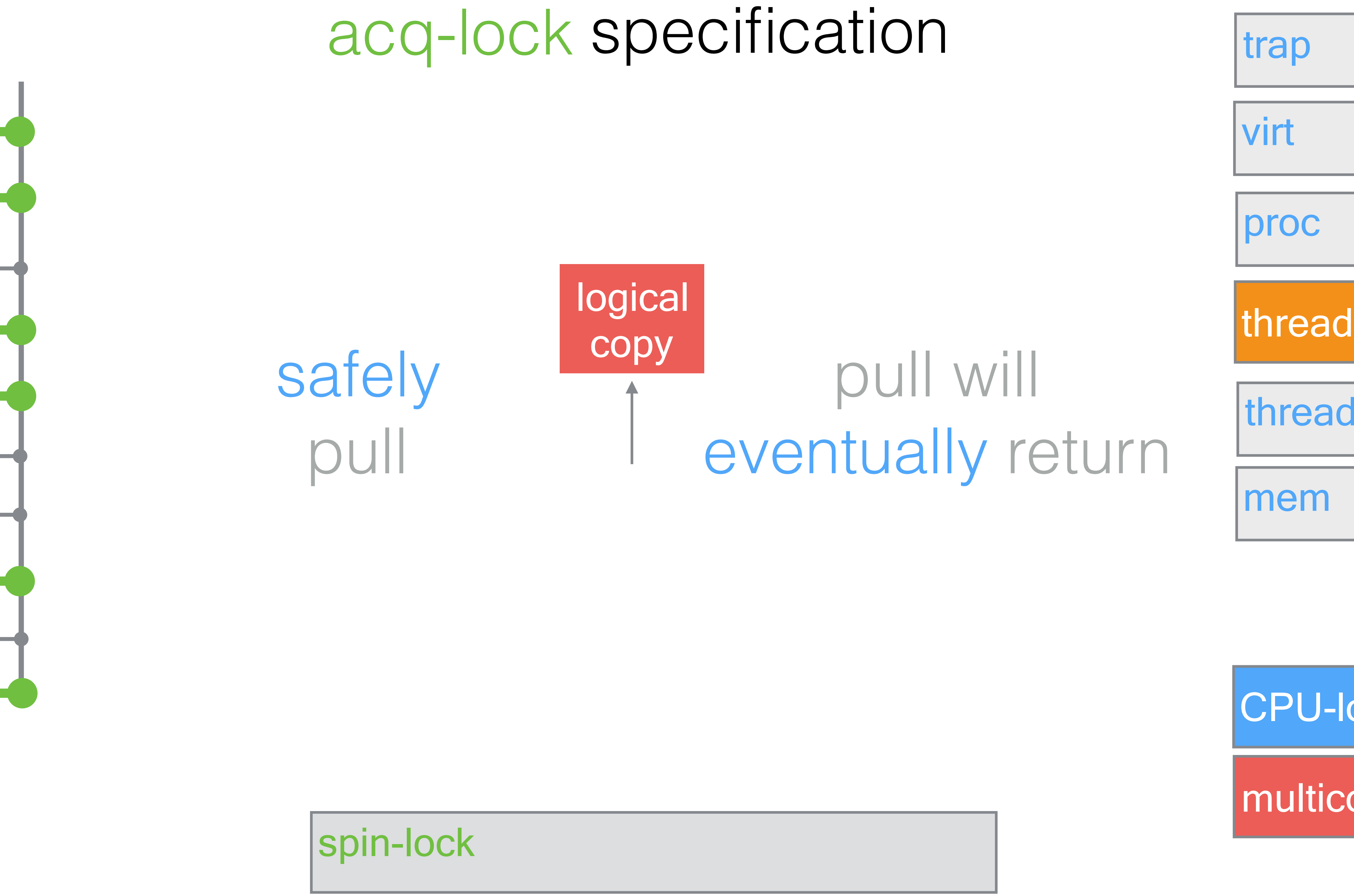

contributions
reuse
atom
0
0.a
1
1
1.a
0
CPU-local machine
CPU i machine
CPU j machine
machine with local copy
$\forall \mathcal{E}_{hs}$
machine with hardware scheduler
multicore machine
trap
virt
proc
thread
thread
mem
spin-lo


trap
virt
proc
thread
thread
mem


trap
virt
proc
thread-local machine
thread
mem
spin-lock
CPU-local machine
multicore machine

acq-lock specification




acq-lock specification
logical copy
safely pull
pull will eventually return
spin-lock
trap
virt
proc
thread
thread
mem
CPU-lo
multico

acq-lock specification

mutual exclusion

logical copy

liveness

spin-lock

trap

virt

proc

thread

thread

mem

CPU-lo

multico

ticket lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint t = FAI_ticket (i
  while (get_now (i) != t)
  { }
  pull (i);
}
```

FAI
ticket

spin-lock

trap

virt

proc

thread

thread

mem

CPU-lo

multic

trap

virt

proc

thread

thread

mem

CPU-lo

multic

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint t = ▶FAI_ticket (i);

  while (▶get_now (i
  { }

  ▶ pull (i);
}
```

get now

FAI ticket

spin-lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint t = FAI_ticket (i);

  while (get_now (i) != t)
  { }

  pull (i);
}
```

get now

FAI ticket

get now

spin-lock

trap

virt

proc

thread

thread

mem

CPU-lo

multico

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint t = ▶FAI_ticket (i);

  while (▶get_now (i) != t)
  { }

  ▶pull (i) pull
}
```

FAI ticket

get now

get now

spin-lock

trap

virt

proc

thread

thread

mem

CPU-lo

multic

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint t = FAI_ticket (i);

  while (get_now (i) != t)
  { }

  pull (i);
}
```

FAI ticket | get now | get now | pull

spin-lock

trap
virt
proc
thread
thread
mem
CPU-lo
multico

mutual exclusion

+ liveness

```
void acq_lock (uint i)
{
  uint t = ▶FAI_ticket (i);

  while (▶get_now (i) != t)
  { }

  ▶pull (i);
}
```

unique t

#CPUs < max_uint

spin-lock

trap

virt

proc

thread

thread

mem

CPU-lo

multico

mutual exclusion + liveness

liveness

```
void acq_lock (uint i)
{
  uint t = ▶FAI_ticket (i);

  while (▶get_now (i) != t)
  { }

  ▶ pull (i);
}
```

FAI ticket | get now | get now | pull

$\underbrace{\text{get now} \quad \text{get now}}_{\text{bounded}}$

#CPUs is bounded

a fair scheduler

lock holders will release lock

spin-lock

trap

virt

proc

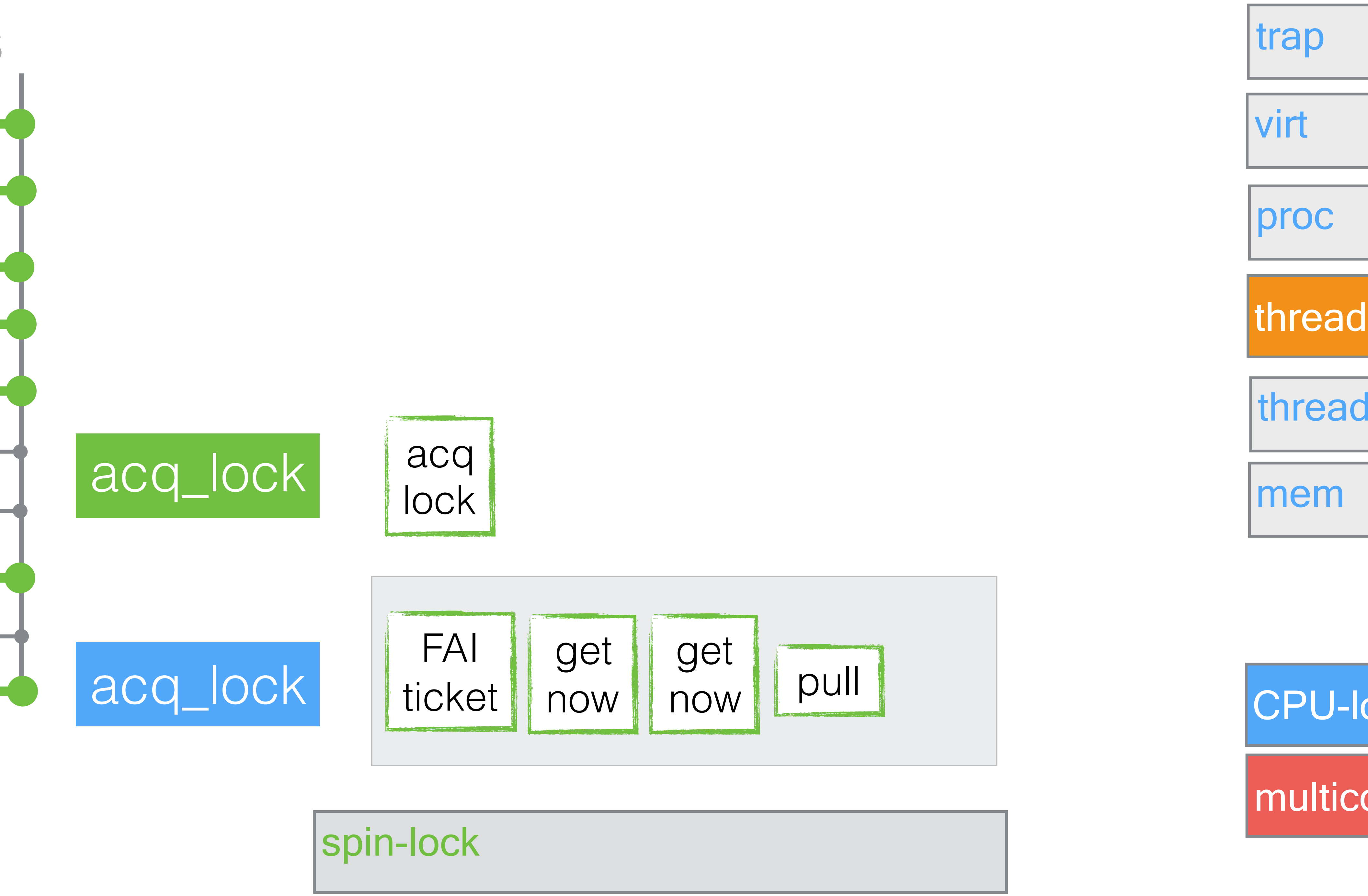
thread

thread

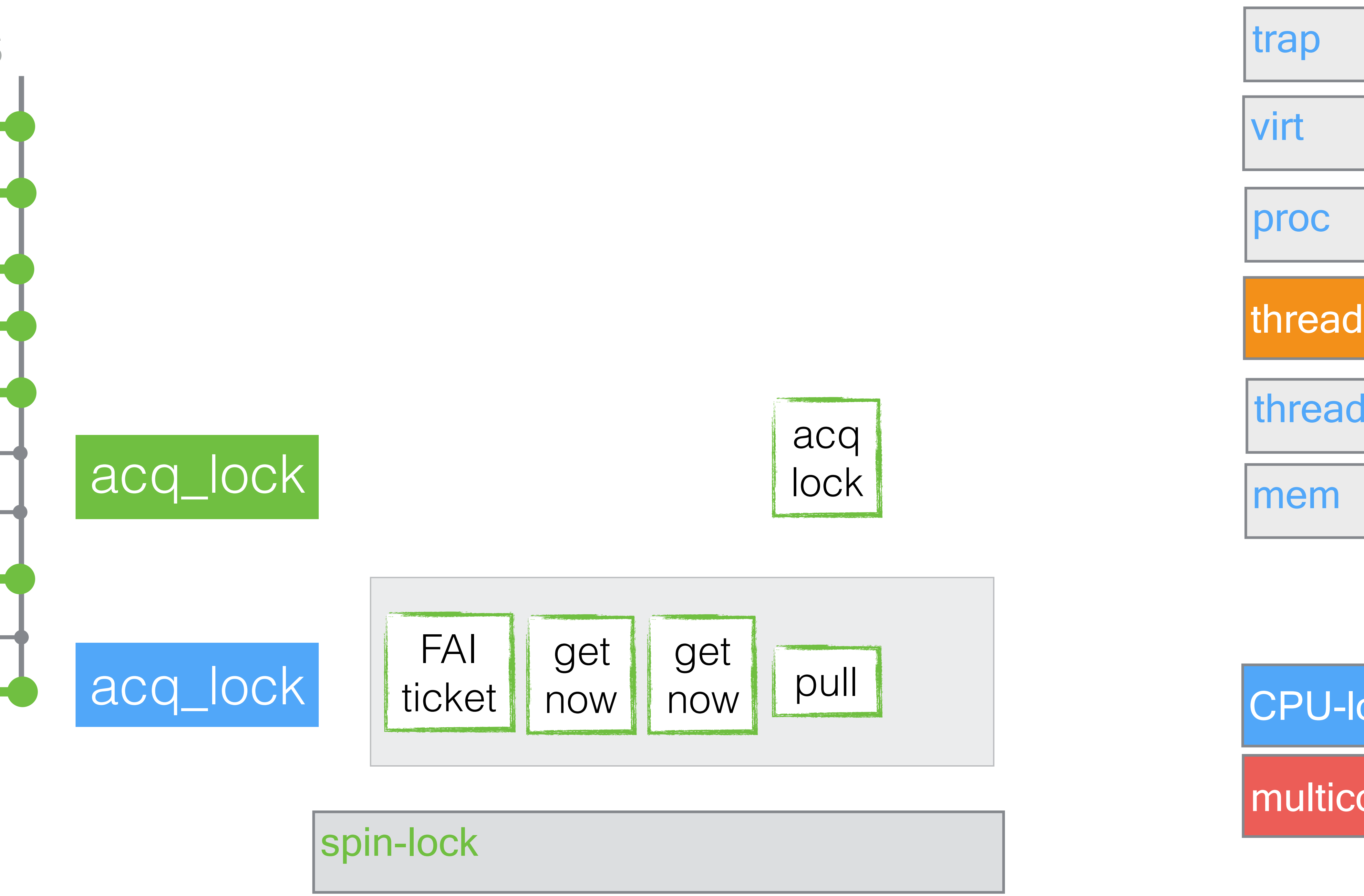
mem

CPU-lo

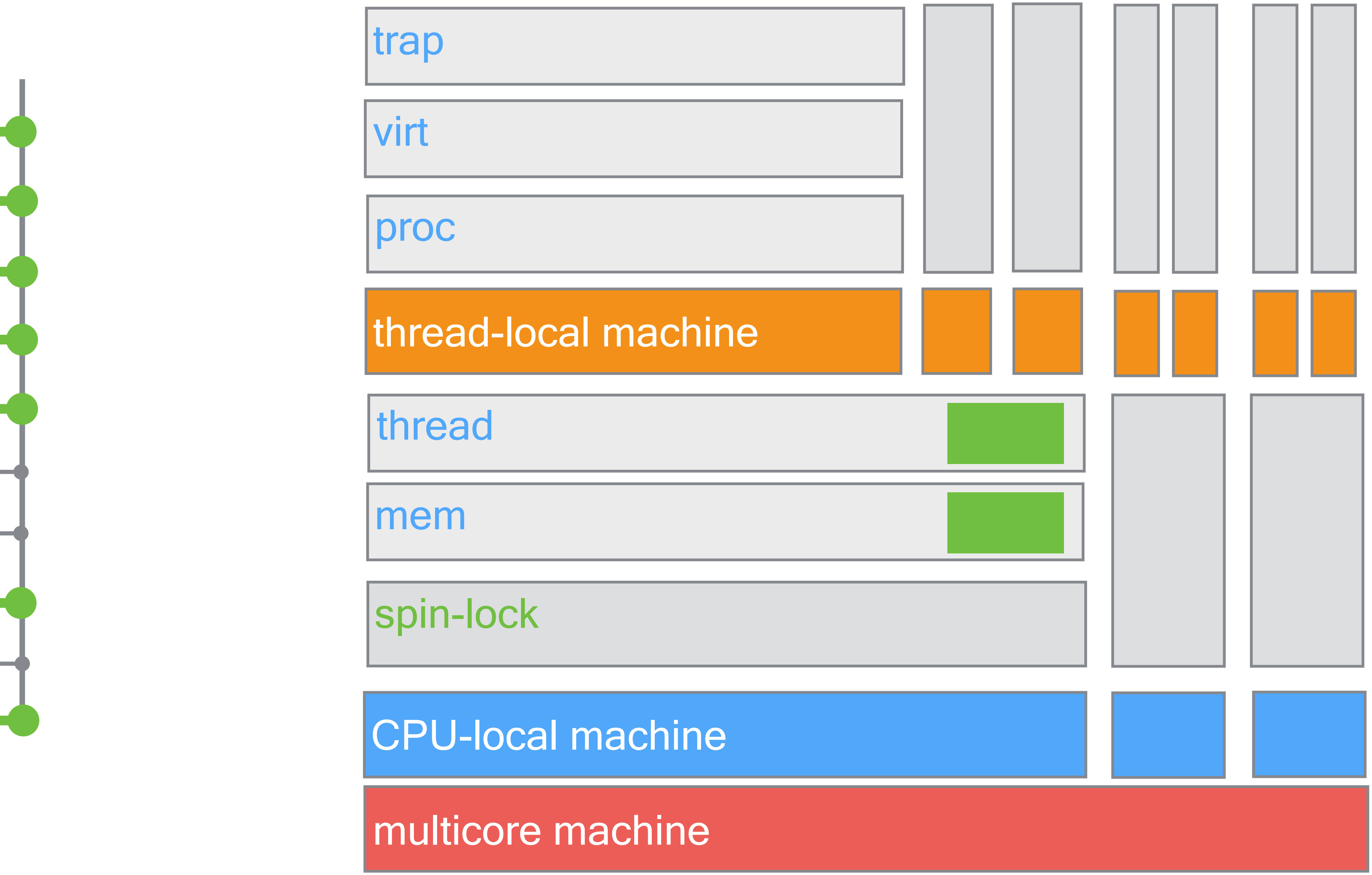
multico

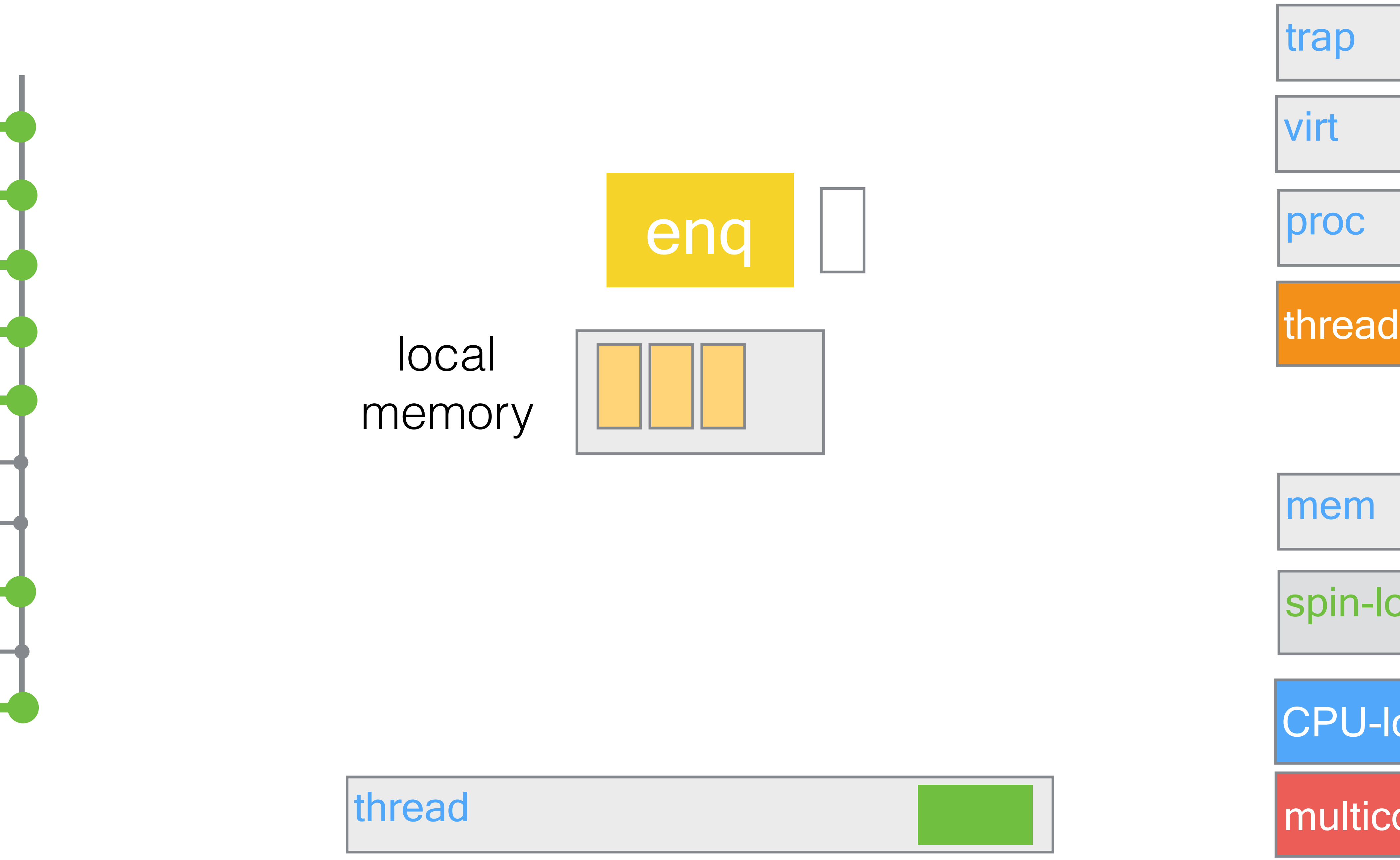


trap
virt
proc
thread
thread
mem
acq_lock
acq
lock
acq_lock
FAI
ticket
get
now
get
now
pull
spin-lock
CPU-lo
multico

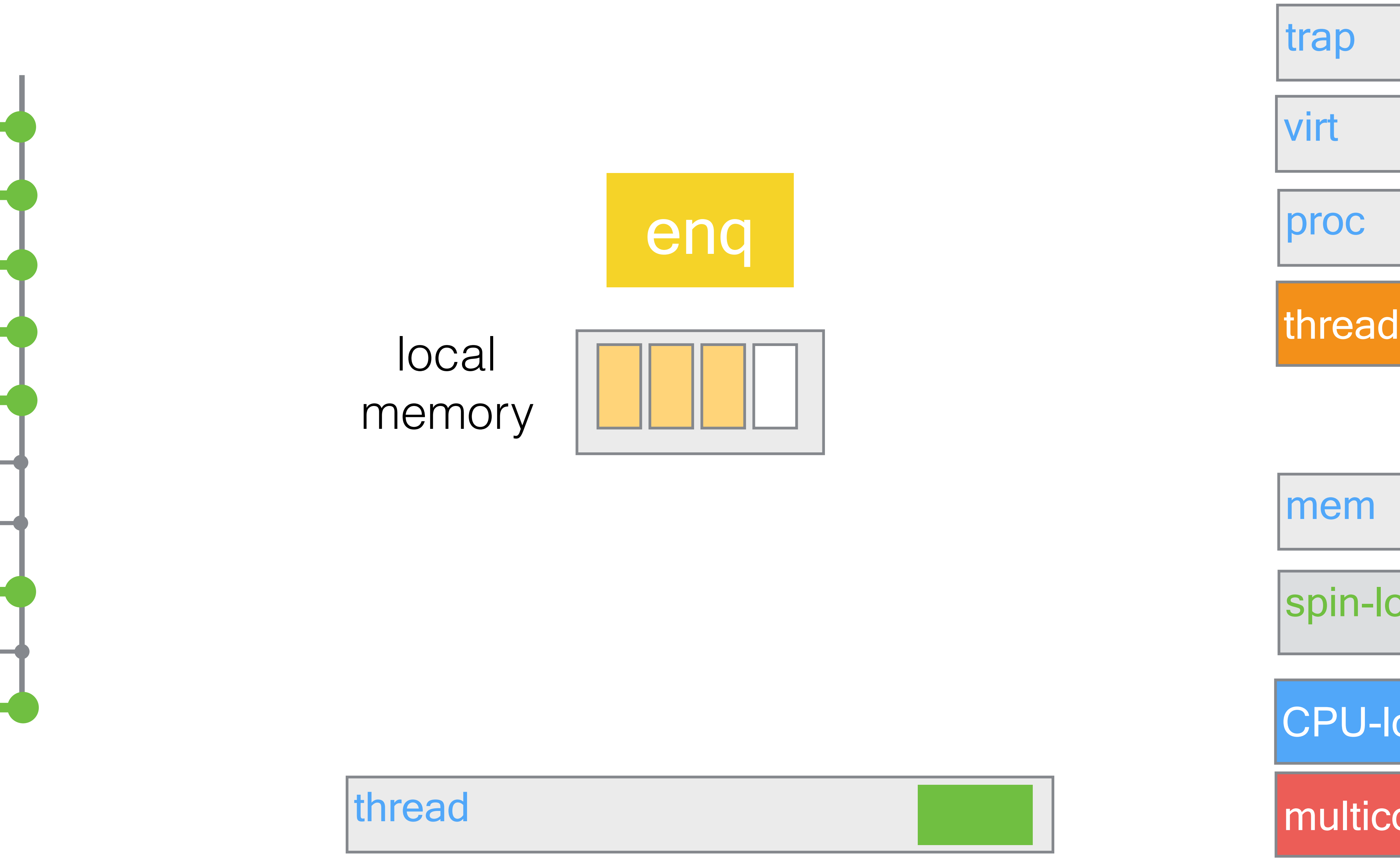

acq_lock
acq_lock
acq
lock
FAI
ticket
get
now
get
now
pull
spin-lock
trap
virt
proc
thread
thread
mem

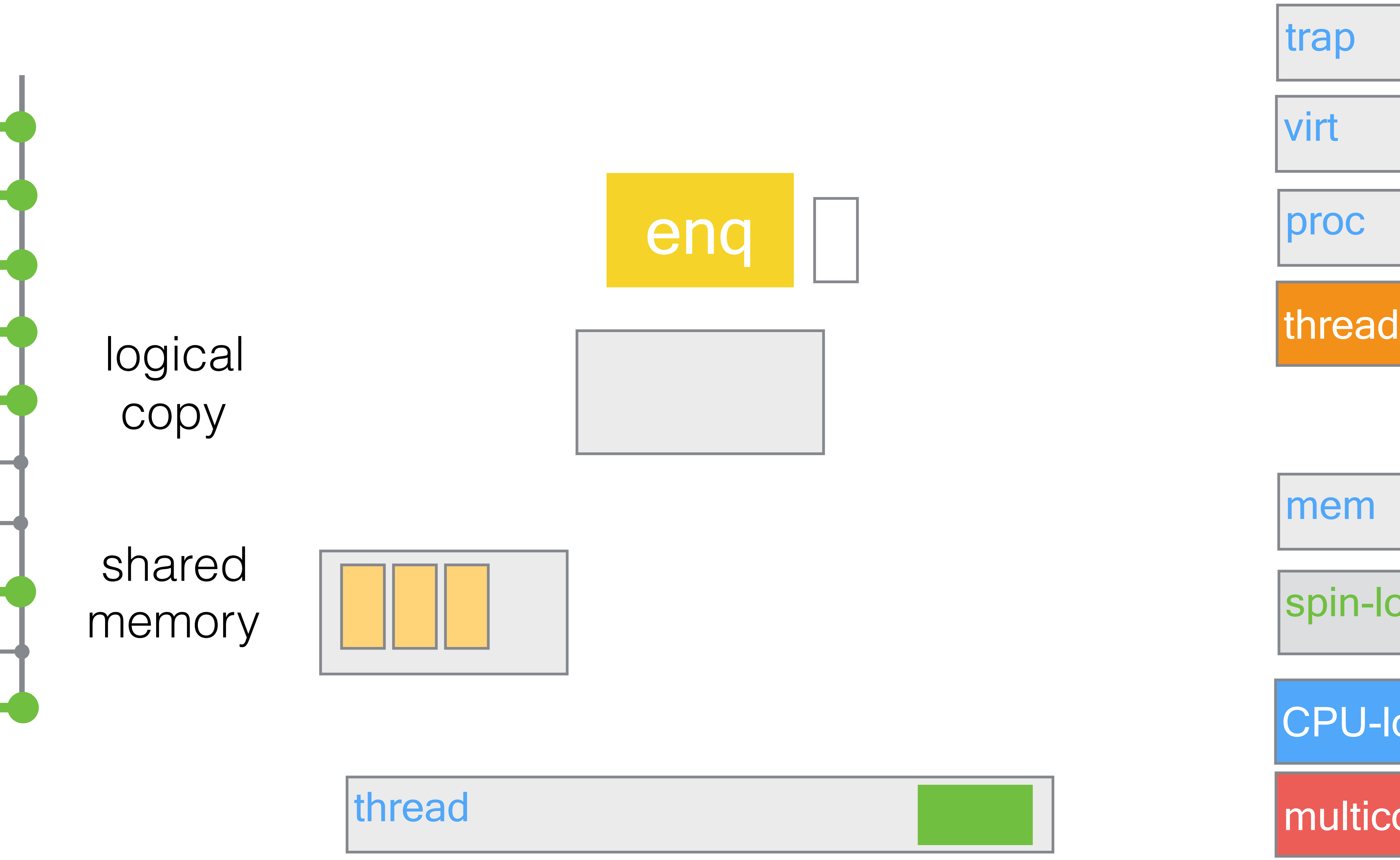

trap
virt
proc
thread
thread
mem
spin-lo
CPU-lo
multic


trap
virt
proc
thread-local machine
thread
mem
spin-lock
CPU-local machine
multicore machine

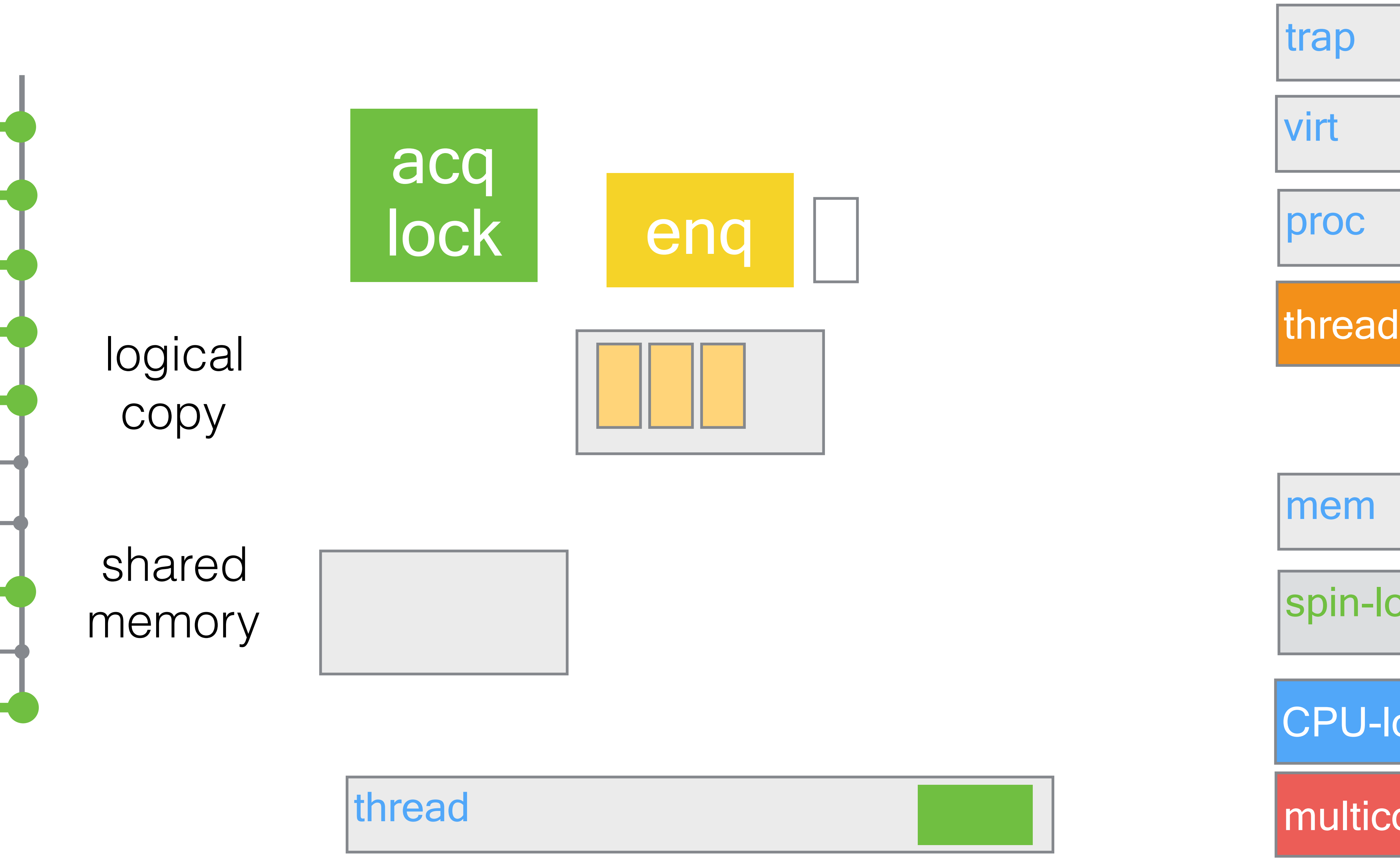


trap
virt
proc
enq
thread
local
memory
mem
spin-lo
CPU-lo
thread
multico

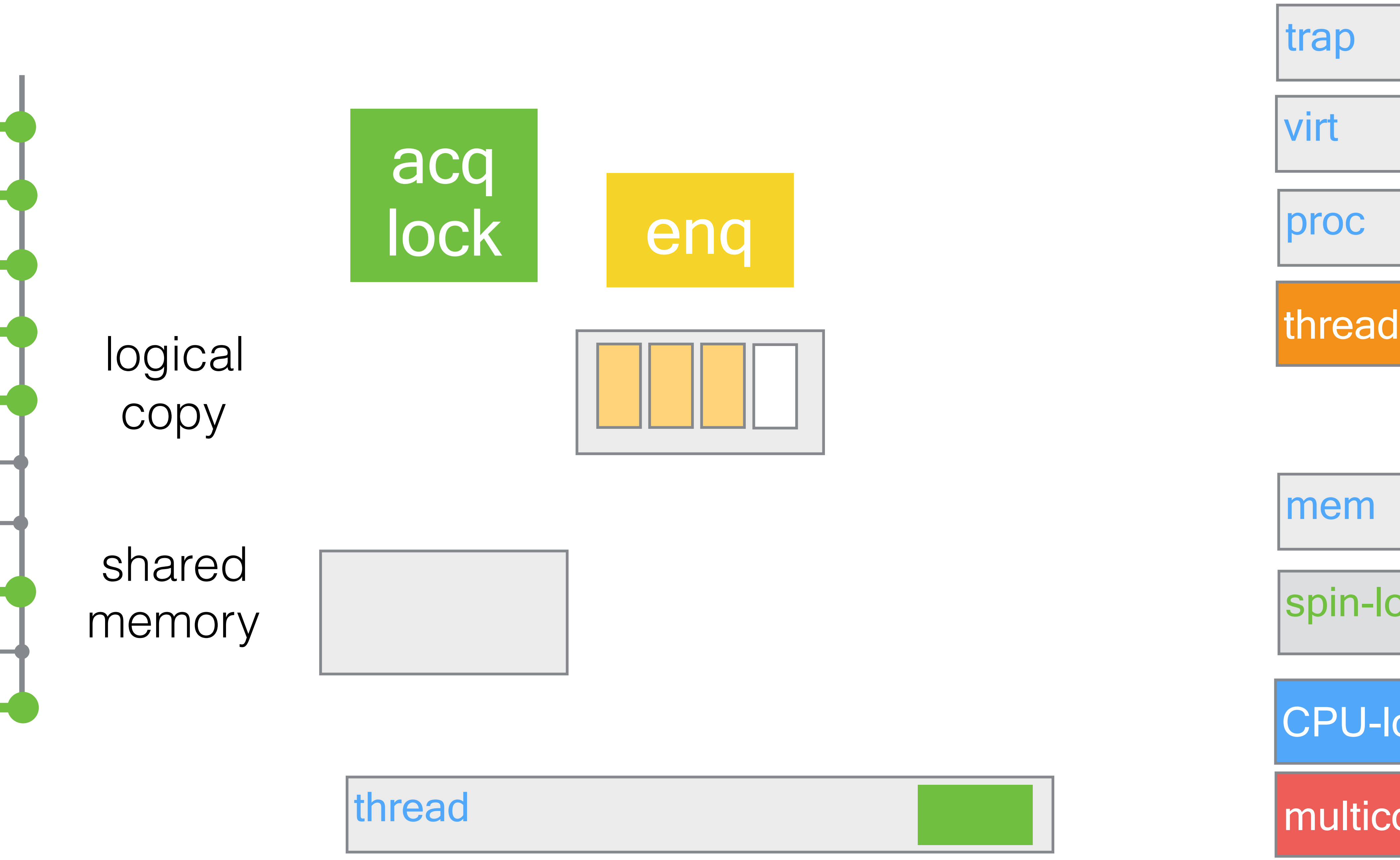

trap
virt
proc
thread
enq
local
memory
mem
spin-lo
CPU-lo
multico
thread

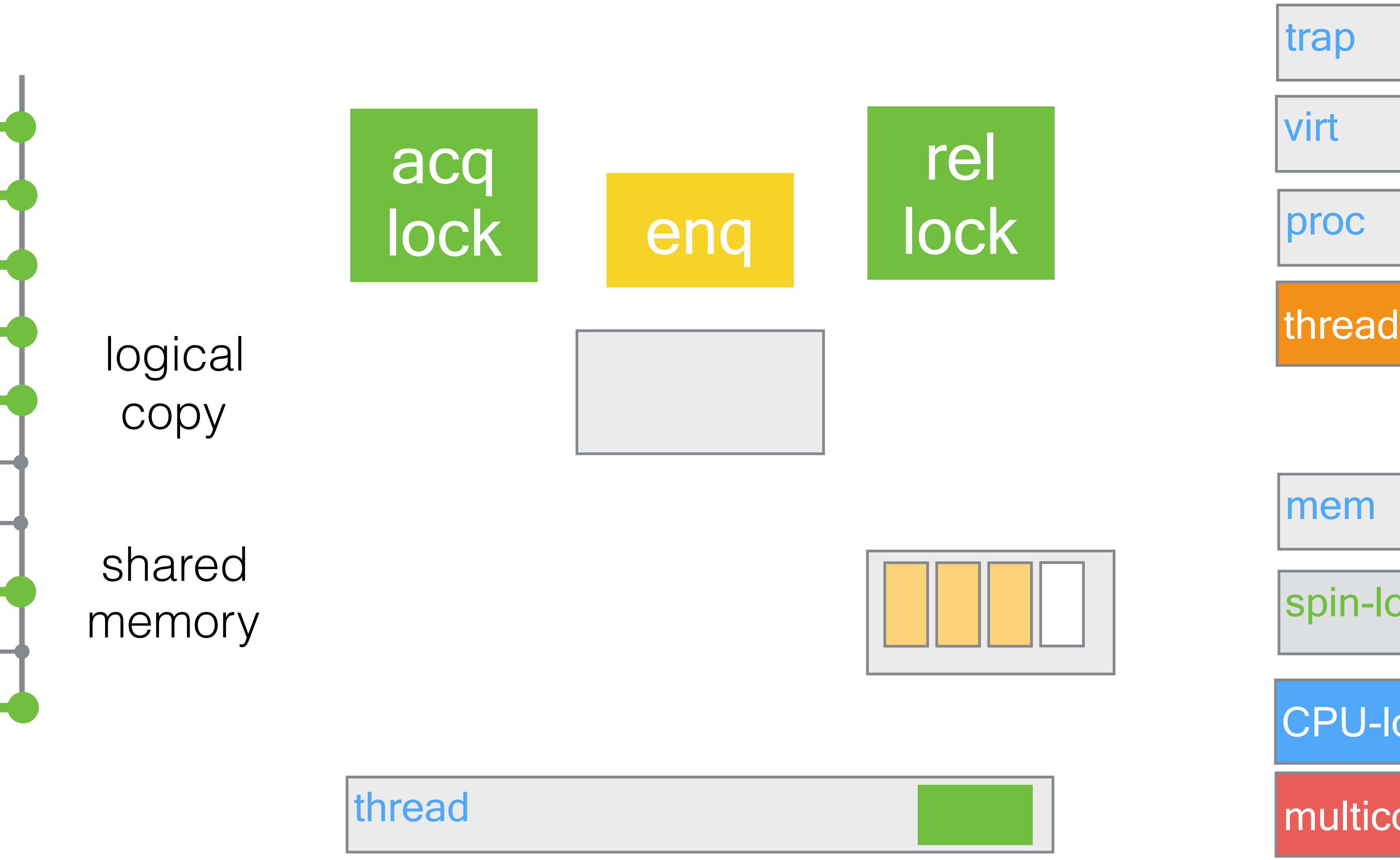


trap
virt
proc
enq
thread
logical
copy
mem
shared
memory
spin-lo
CPU-lo
thread
multico

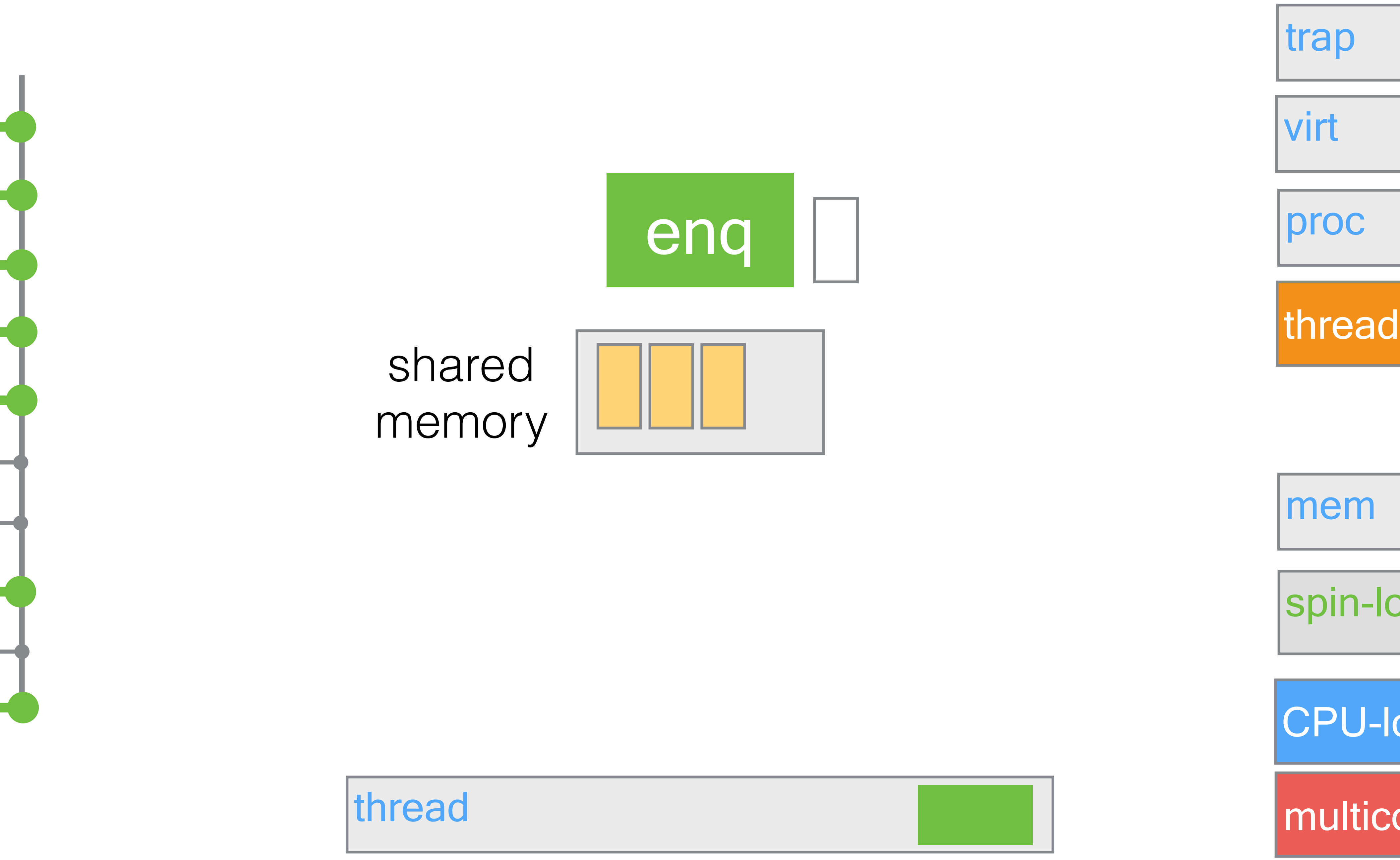


acq
lock
enq
logical
copy
shared
memory
thread
trap
virt
proc
thread
mem

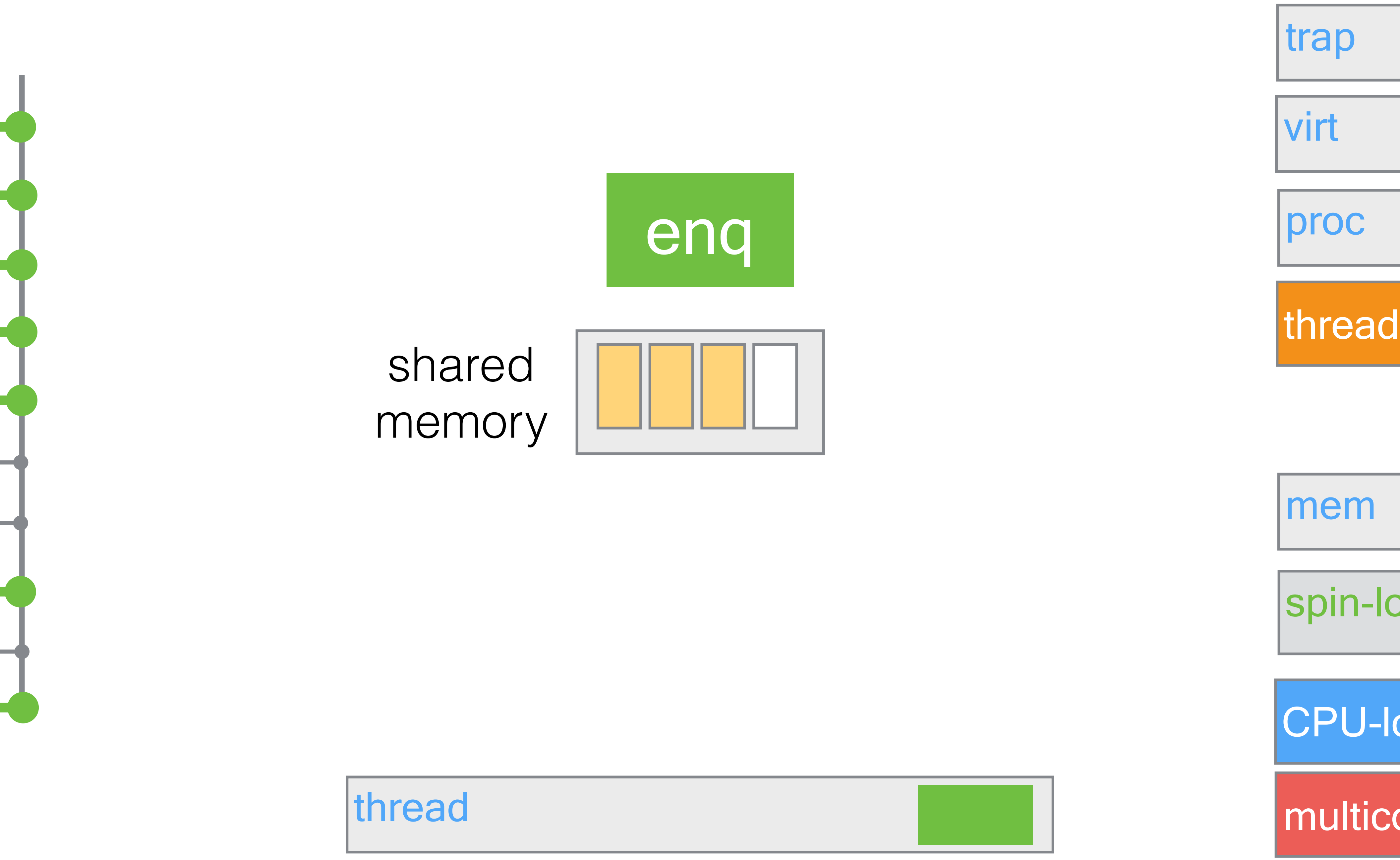

acq
lock
enq
logical
copy
shared
memory
thread
trap
virt
proc
thread
mem
spin-lo
CPU-lo
multico



acq lock
enq
rel lock
logical copy
shared memory
thread
trap
virt
proc
thread
mem
spin-lo
CPU-lo
multico



enq
shared memory
thread
trap
virt
proc
thread
mem
spin-lo
CPU-lo
multic


trap
virt
proc
enq
thread
shared
memory
mem
spin-lo
CPU-lo
thread
multic

trap
virt
proc
thread
thread
mem
spin-lo
CPU-lo
multico

contributions

```
void yield ()
{
  uint t =  tid();
  ...
  ▶enq (t, rdq());

  uint s = ▶deq (rdq());
  ...
  context_switch (t, s)
}
```

thread-local machine

trap

virt

proc

thread

mem

spin-lo

CPU-lo

multico

contributions

asm&C

CompcertX

```
void yield ()
{
  uint t =  tid();
  ...
  enq (t, rdq());

  uint s = deq (rdq());
context_switch s)
}
```

thread-local machine

trap

virt

proc

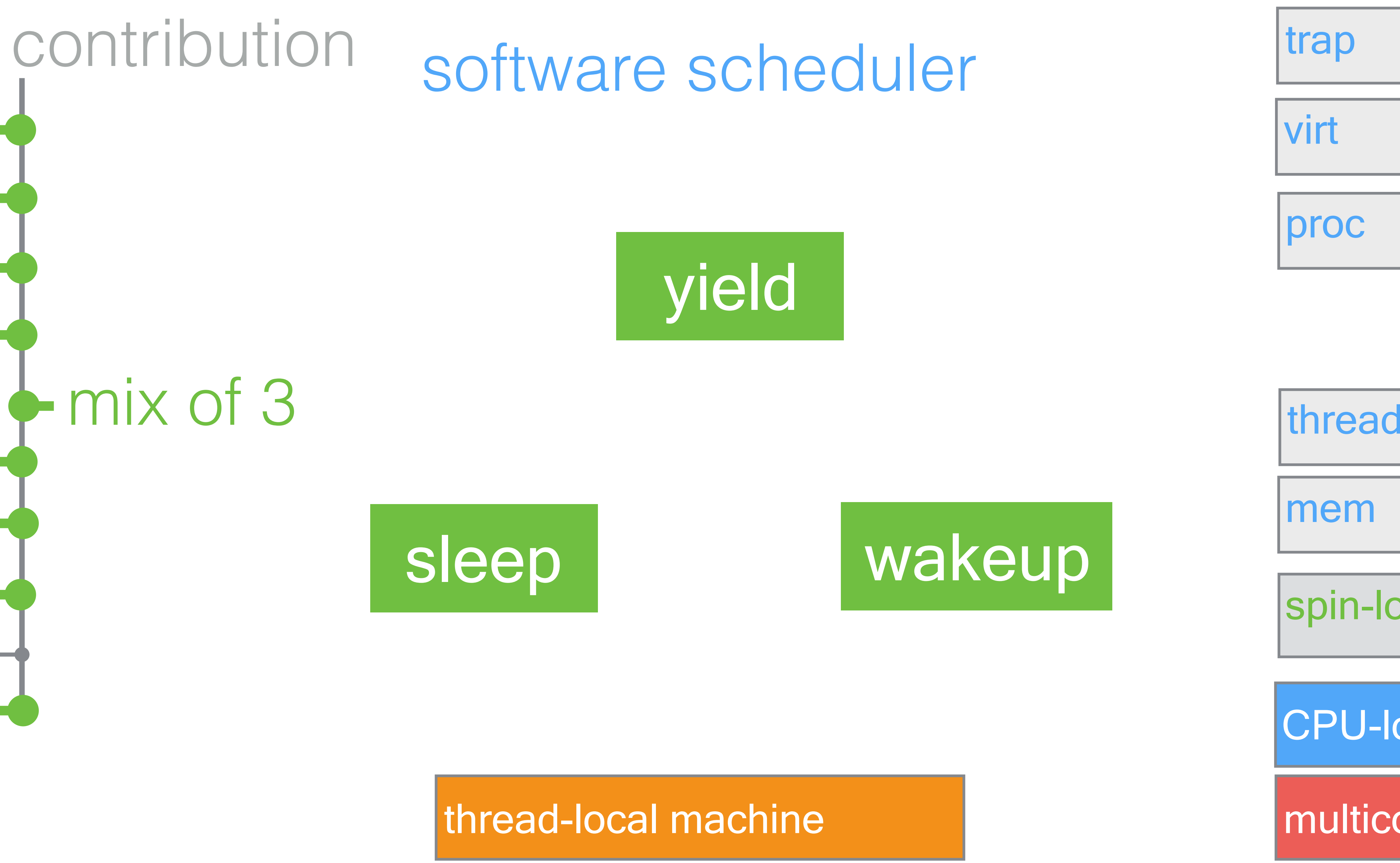
thread

mem

spin-lo

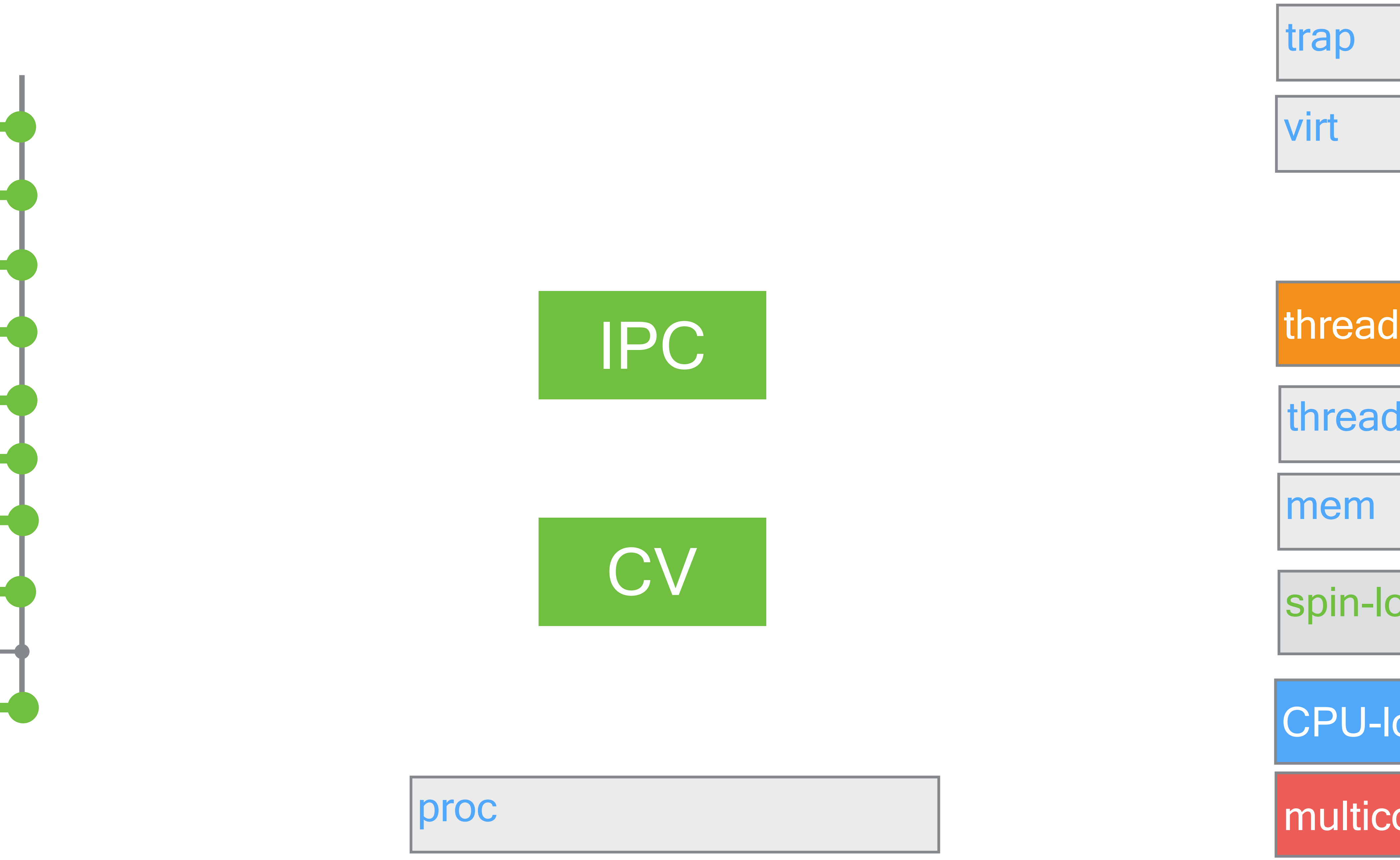
CPU-lo

multico



contribution
software scheduler
mix of 3
yield
sleep
wakeup
thread-local machine
trap
virt
proc
thread
mem
spin-lo
CPU-lo
multic


trap
virt
proc
thread
thread
mem



trap
virt
IPC
thread
thread
mem
CV
spin-lo
CPU-lo
proc
multic

evaluation:

proof effort for concurrency(LOC)

top spec: 450

machine model: 943

intermediate spec: 40K

proof(concurrency): 50K

Coq &machine checkable

2 person year

trap

virt

proc

thread

thread

mem

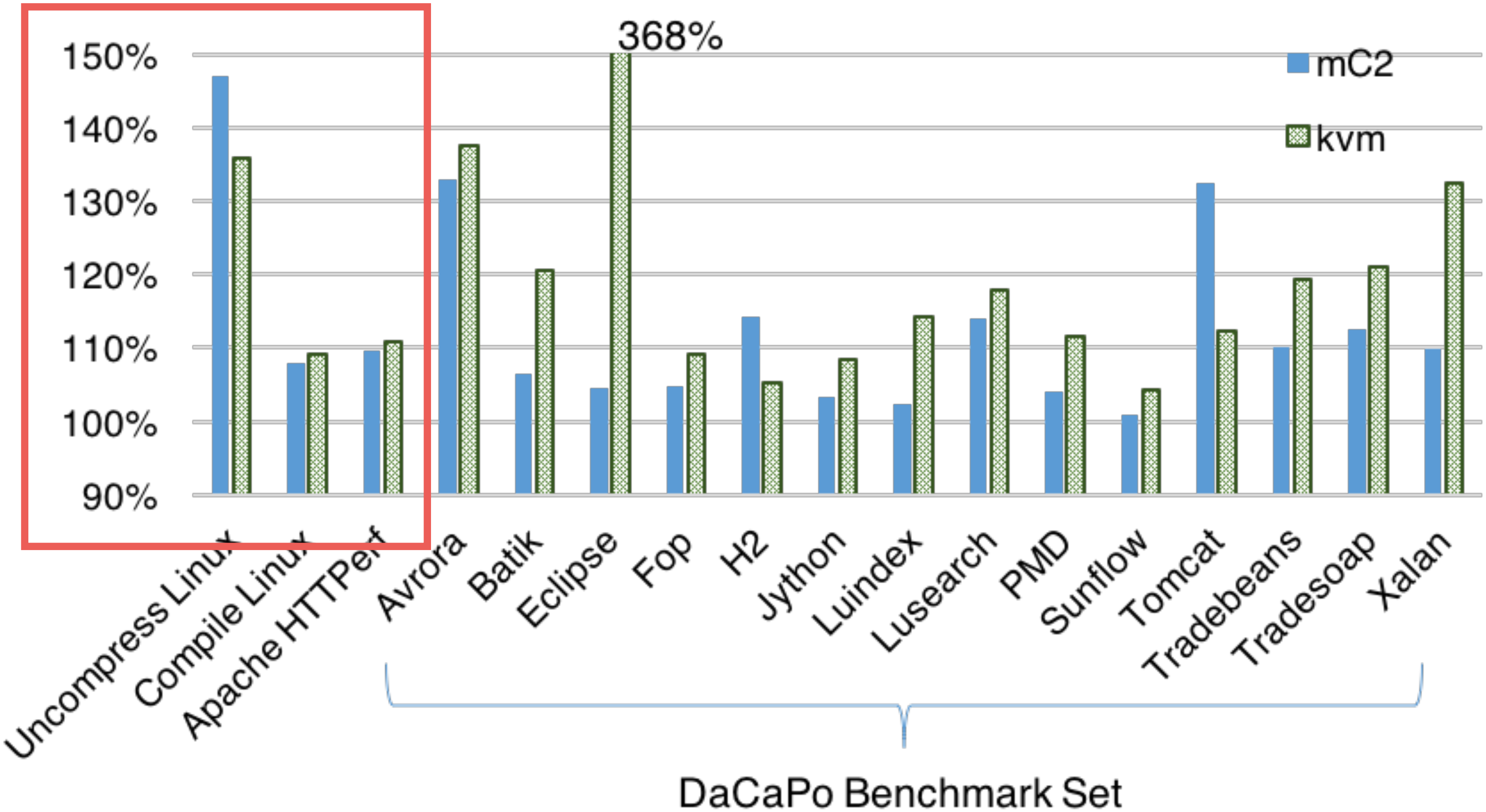
spin-lo

CPU-lo

multico

evaluation: performance

mC2 is comparable with kvm



trap

virt

proc

thread

thread

mem

spin-lo

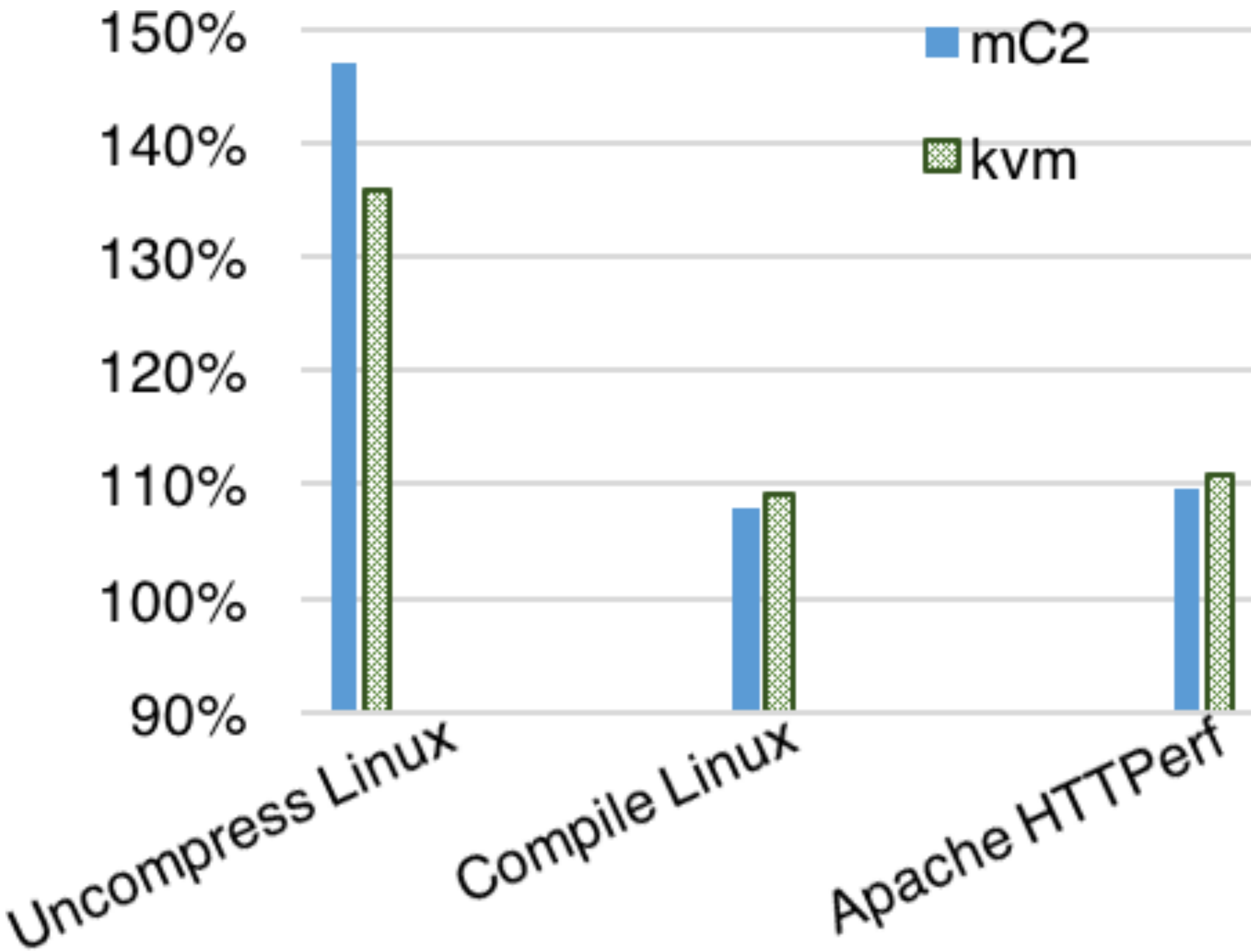
CPU-lo

multico

evaluation:

performance

mC2 is comparable with kvm



trap

virt

proc

thread

thread

mem

spin-lo

CPU-lo

multico

limitations & future work

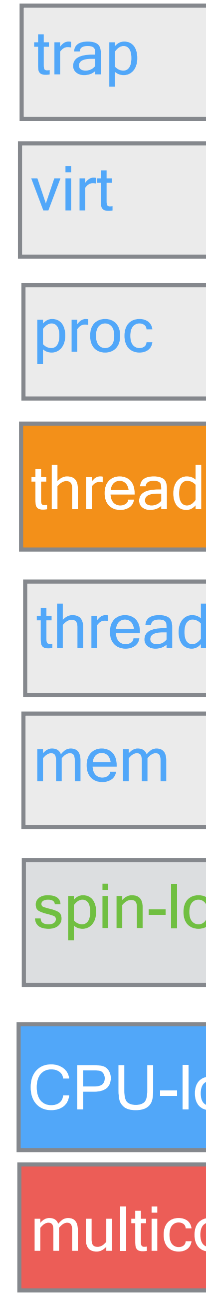
bootloader

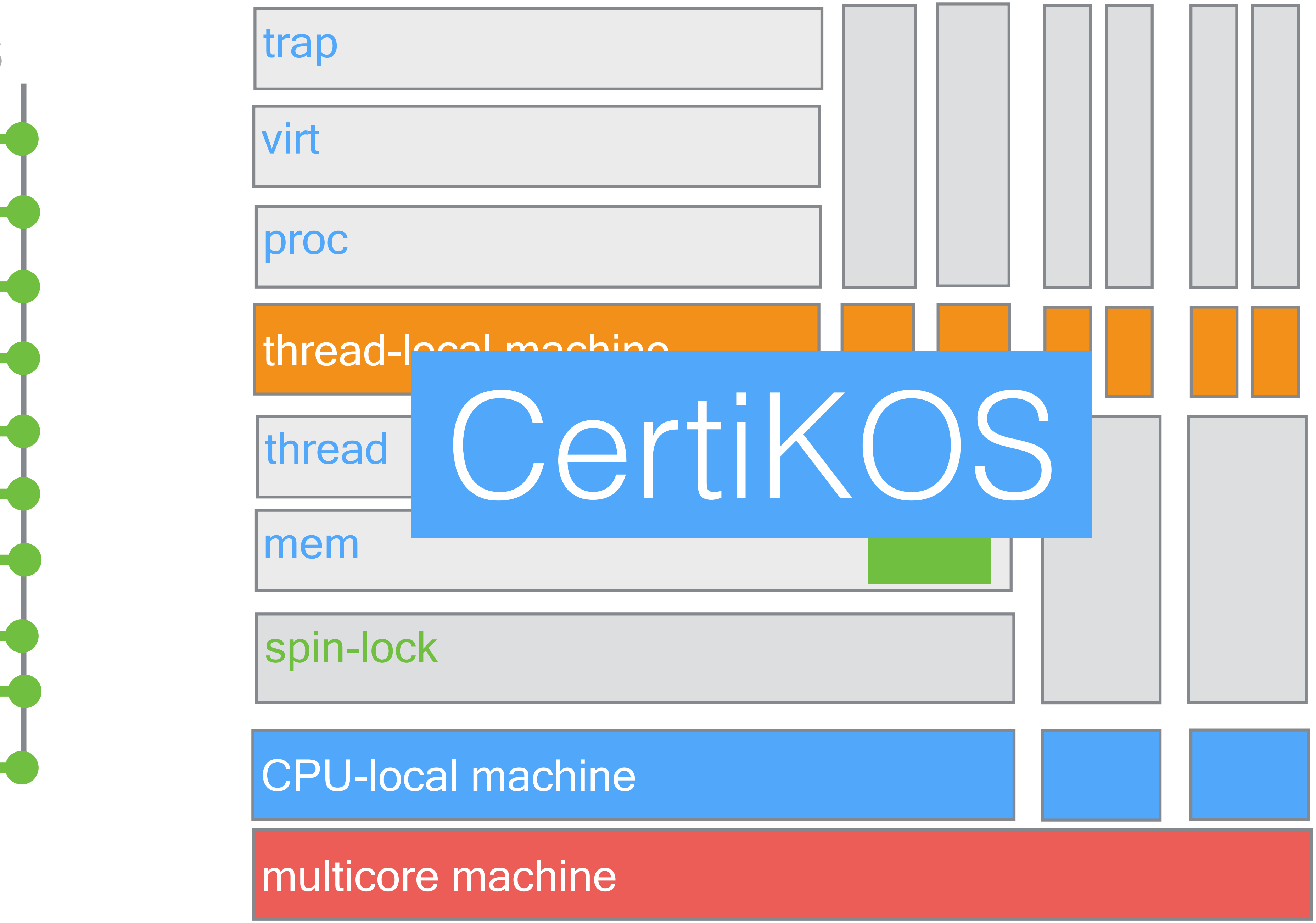
assembler of CompCert

machine model is in the TCB

sequential consistency

file system & network stack




trap
virt
proc
thread-local machine
CertiKOS
thread
mem
spin-lock
CPU-local machine
multicore machine

contributions

- mC2
- fine-grained lock
- liveness
- reuse
- mix of 3
- asm&C
- CompcertX
- extensibility
- Coq &machine checkable
- 2 person year

CertiKOS

trap

virt

proc

thread

thread

mem

spin-lo

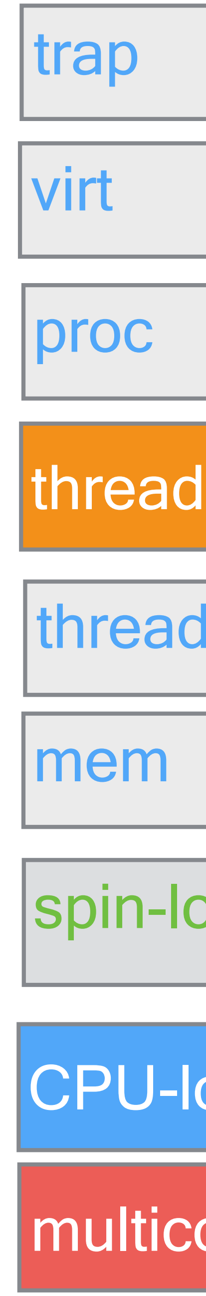
CPU-lo

multico

CertiKOS

mC2

the first formally verified concurrent OS kernel.



CertiKOS

new technical contributions

certified concurrent layers

logical log + hardware scheduler
+ environment context

push/pull model

multicore machine lifting