



# Threading in the VM

---

- Design depends on:
  - Needed interaction with OS
  - Needed interaction with external code
  - Needed CPU scalability
  - Simplicity
  - Language features



# Types of Threading in VM

---

- Internal thread scheduling
  - Only one thread runs at a time
- External thread scheduling
  - Native scheduling
- Mixture thread scheduling
  - External scheduling of native code
  - Internal scheduling of OO code



# Internal Thread Scheduling

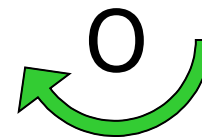
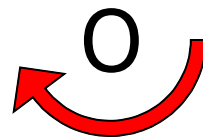
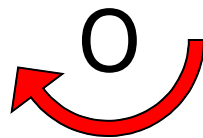
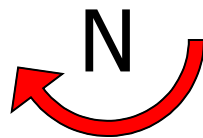
---

- Only one native thread is used
- VM does the scheduling of OO threads
- Ideal for embedded systems without real OS
- Very popular before OSs had threads.

Threads:

N – in native\*

O – in OO



← Current thread



# Internal Thread Scheduling

---

- Pros:
  - Very simple implementation
  - Can have very many internal threads
- Cons:
  - Cannot handle blocking OS calls
  - Is not CPU scalable
  - Does not work well with legacy code



# Blocking and Internal Thread Scheduling

---

- The VM cannot schedule threads stuck in external code
- Therefore it is forbidden to call blocking external code. On Unix use `fcntl` `O_NONBLOCK`
- `read()` etc. return `EAGAIN`
- `select()` wakes up VM



# External Thread Scheduling

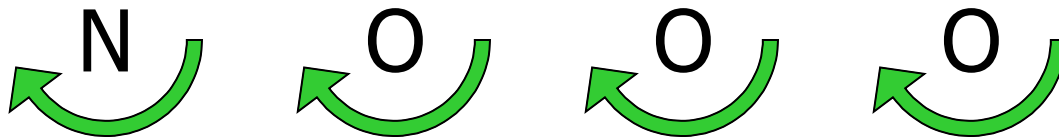
---

- One native thread per OO thread
- The OS schedules the OO threads
- Used where CPU scalability is needed and interfacing to legacy code is important

Threads:

N – in native

O – in OO





# External Thread Scheduling

---

- Pros:
  - Interfaces well with OS and legacy code
  - Scales as well as the underlying OS
- Cons:
  - Introduces MT-issues in VM
  - Complicated safe-point roll forward



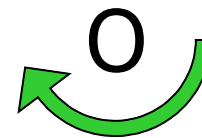
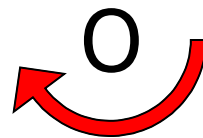
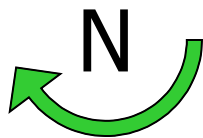
# Mixed Thread Scheduling

---

- One native thread per OO thread
- The OS schedules the OO threads if they are running native code
- The VM schedules the OO threads if they are running OO code

Threads:

N – in native  
O – in OO



← Current thread



# Mixed Thread Implementation

---

- Big mutex around OO execution
- This is the V8 model
  - Uses explicit unlocking in external calls
  - Programmer must decide whether an external call is long-running or not
  - Some overhead in unlocking for fast external calls



# Mixed Thread Scheduling

---

- Pros:
  - Interfaces well with OS and legacy code
  - Simple implementation
- Cons:
  - No CPU scalability in OO part



# Lazy Mixed Thread Implementation

---

- Pros:
  - Interfaces well with OS and legacy code
  - Simple implementation
  - Can have many internal threads
- Cons
  - No CPU scalability in OO part



# Lazy Mixed Threads Implementation

---

- Variant on Mixed Thread Implementation
  - Used in OOVM
  - If a timer tick occurs while we are in external code then take Mutex away from thread and start up another thread
  - Short external calls have no overhead
  - We only need as many OS threads as there are blocking external calls outstanding