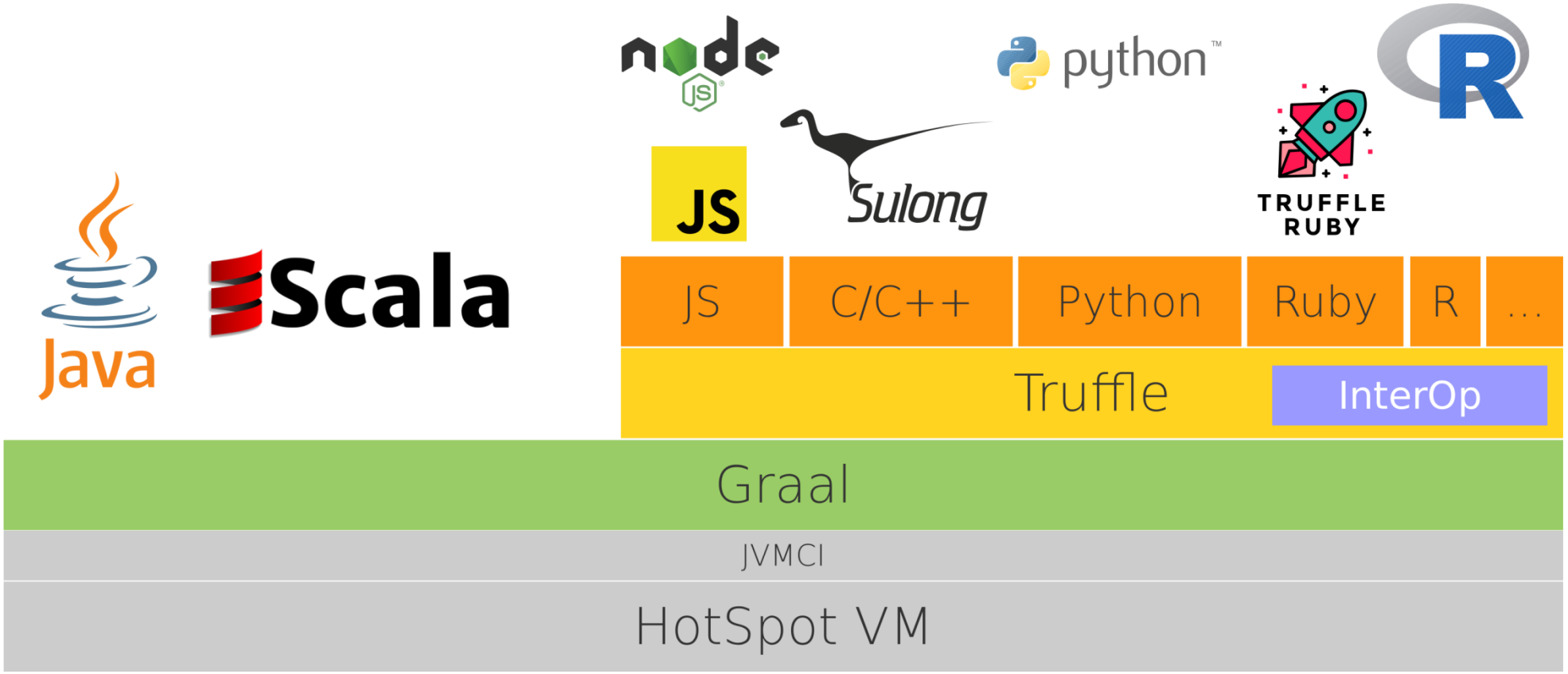
Substrate VM

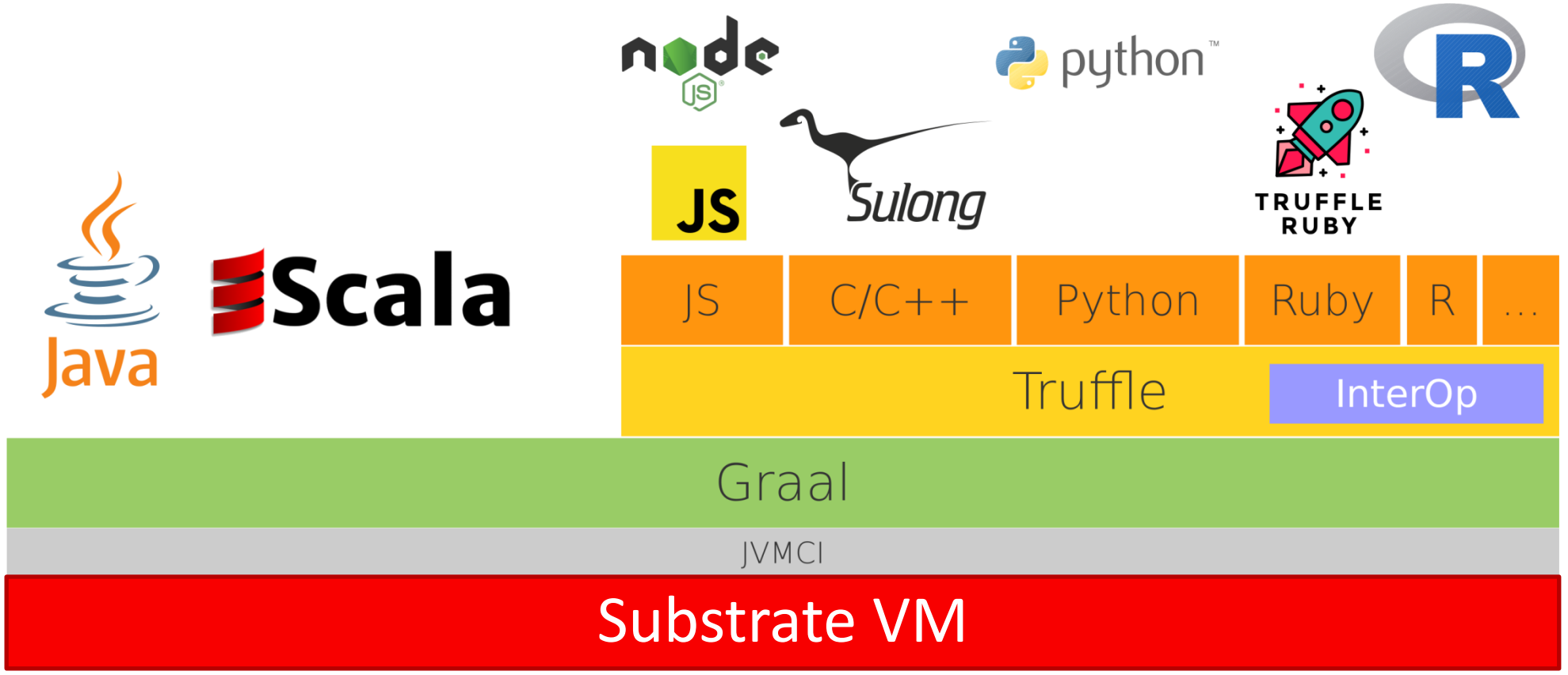
Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

GraalVM Stack



GraalVM Stack



Substrate VM is …

… an **embeddable** VM

with **fast startup** and **low footprint**

for, and written in, a **subset of Java**

optimized to **execute Truffle** languages

ahead-of-time compiled using Graal

integrating with **native development tools.**

Substrate VM: Execution Model

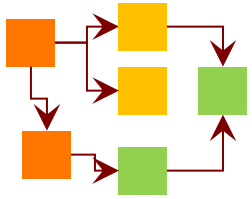
Points-To Analysis

Ahead-of-Time Compilation

Truffle Language

JDK

Substrate VM



Machine Code

Initial Heap

DWARF Info

ELF / MachO Binary

All Java classes from Truffle language (or any application), JDK, and Substrate VM

Reachable methods, fields, and classes

Application running without dependency on JDK and without Java class loading

Features of Substrate VM

- Type safety and memory safety of Java
  - Type checks, array bounds checks, null pointer checks
- Garbage collection
  - All Java memory is managed automatically
- JDK support
  - Most core and utility classes
- C code integration
  - SystemJava: access C functions and C data structures without performance overhead
- Multithreading (optional feature)
  - Everything in `java.util.concurrent` package
- Native tool support for debugging, profiling, ...
  - Standard DWARF debug information for ahead-of-time compiled code and dynamically compiled code

"Hello World" in C, Java, Ruby

Language	Virtual Machine	Instructions	Time	
C hello		147,000	< 1 ms	puts("Hello!");
Java	Java HotSpot VM 1.8.0_121-b13	140,000,000	50 ms	System.out.println("Hello! ")
Java	GraalVM 0.21 Substrate VM	332,000	< 1 ms	System.out.println("Hello! ")
Ruby	MRI 2.3.3p222	125,000,000	35 ms	ruby -e 'puts "Hello!"'
Ruby	GraalVM 0.21 truffleruby	40,754,085,818	4955 ms	ruby -e 'puts "Hello!"'
Ruby	GraalVM 0.21 Substrate VM truffleruby image	345,160,332	90 ms	ruby -e 'puts "Hello!"'

Substrate VM gets Ruby startup close to MRI

Operating system: Linux x86_64
Instructions: perf stat -e instructions ...
Time: time (bash builtin) ...

SystemJava: Integrate Java and C Code

```
@CFunction static native int clock_gettime(int clock_id, timespec tp);
```

```
int clock_gettime(clockid_t __clock_id, struct timespec *__tp)
```

```
@CConstant static native int CLOCK_MONOTONIC();
```

```
#define CLOCK_MONOTONIC 1
```

```
@CStruct interface timespec extends PointerBase {
  @CField long tv_sec();
  @CField long tv_nsec();
}
```

```
struct timespec {
  __time_t tv_sec;
  __syscall_slong_t tv_nsec;
};
```

```
@CPointerTo(nameOfCType="int") interface CIntPointer extends PointerBase {
  int read();
  void write(int value);
}
```

```
int* pint;
```

```
@CPointerTo(CIntPointer.class) interface CIntPointerPointer ...
```

```
int** ppint;
```

Implementation of `System.nanoTime()` using SystemJava:

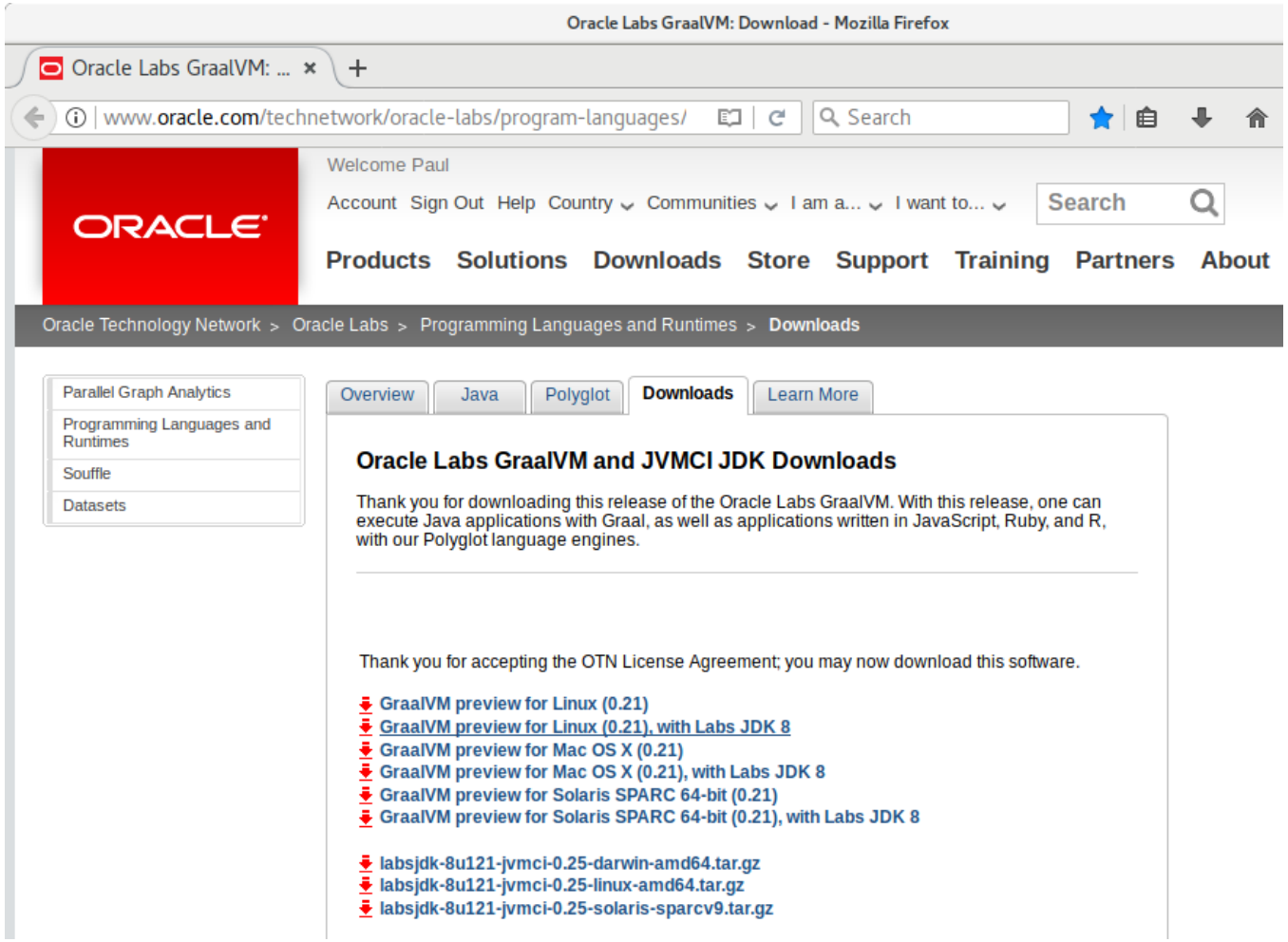
```
static long nanoTime() {
  timespec tp = StackValue.get(SizeOf.get(timespec.class));
  clock_gettime(CLOCK_MONOTONIC(), tp);
  return tp.tv_sec() * 1_000_000_000L + tp.tv_nsec();
}
```

Dependencies

- Standard C library
  - Allocate and free memory
  - Low-level file access
  - Optional: pthread
- No dependency on
  - Signal handling
  - Memory protection
  - Standard C++ library
  - Any Java virtual machine, Java bytecodes
  - Any native library of the JDK

- Example: all dependencies of the JavaScript executable
  - Memory mapping
    - mmap64, munmap
  - C Memory allocation
    - malloc, calloc, free
  - Core library
    - memmove, exit, gettimeofday, clock_gettime, strerror, __errno_location, getuid, getpwuid, sysconf
  - File access
    - open, close, read, write, pread64, opendir, closedir, readdir64_r, lseek, realpath, getcwd, dup2, fsync, ioctl, fcntl, unlink, __xstat64, __fxstat64
  - Thread
    - pthread_create, pthread_join, sched_yield, pthread_key_create, pthread_mutex_init, pthread_cond_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_wait, pthread_cond_timedwait, pthread_cond_signal, pthread_cond_broadcast, pthread_getspecific, pthread_setspecific, pthread_cancel, pthread_testcancel, pthread_attr_init, pthread_attr_setguardsize, pthread_attr_setstacksize, pthread_attr_destroy

Try It Yourself:



http://www.oracle.com/technetwork/oracle-labs/program-languages/downloads/index.html

Integrated Cloud

Applications & Platform Services


ORACLE®