Inside Virtual Machines

Jonathan Worthington

Scarborough Linux User Group

Introduction

Inside Virtual Machines

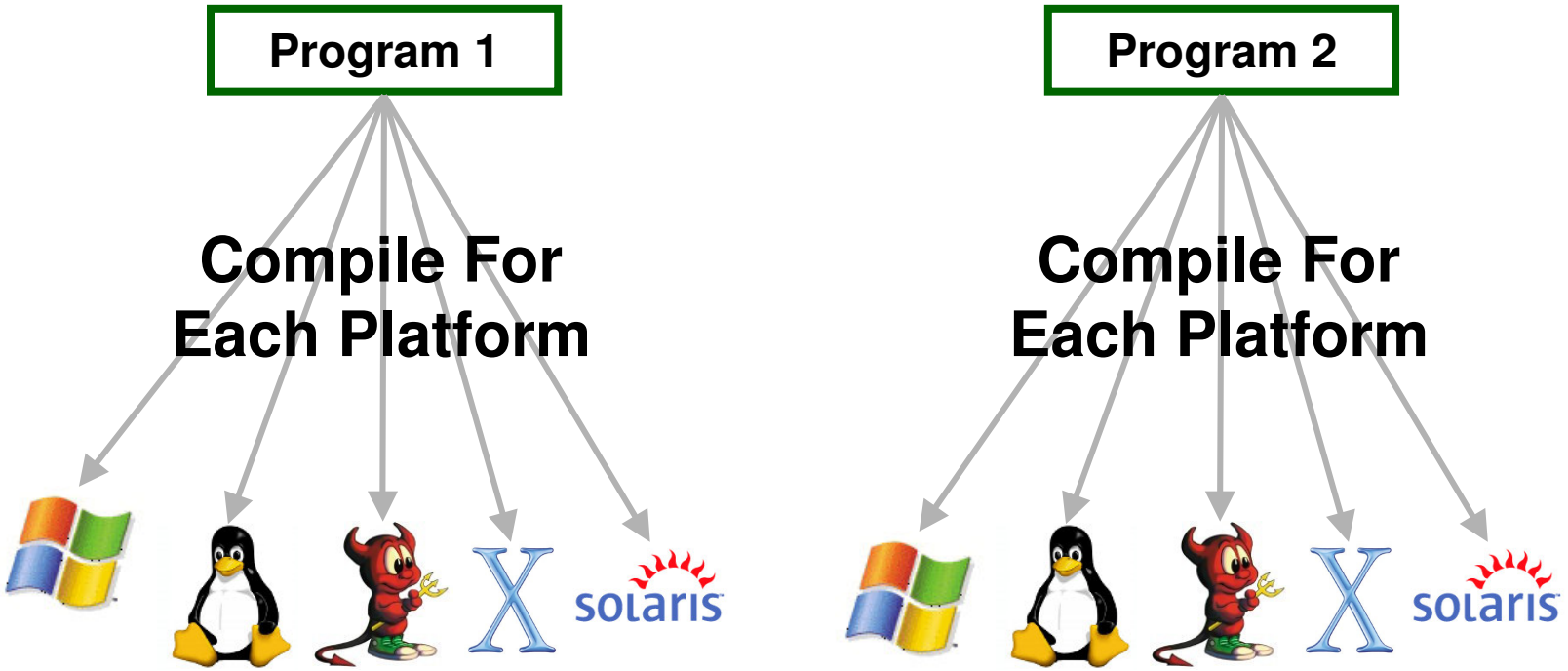
What does a Virtual Machine do?

- Hides away the details of the hardware platform and operating system.
- Defines a common set of instructions.
- Abstracts away operating system details
- Efficiently translates the virtual instructions to those supported by the hardware CPU.
- Provides support for high level language constructs (such as subroutines, OOP).

Inside Virtual Machines

Why Virtual Machines?

1. Simplified software development and deployment.

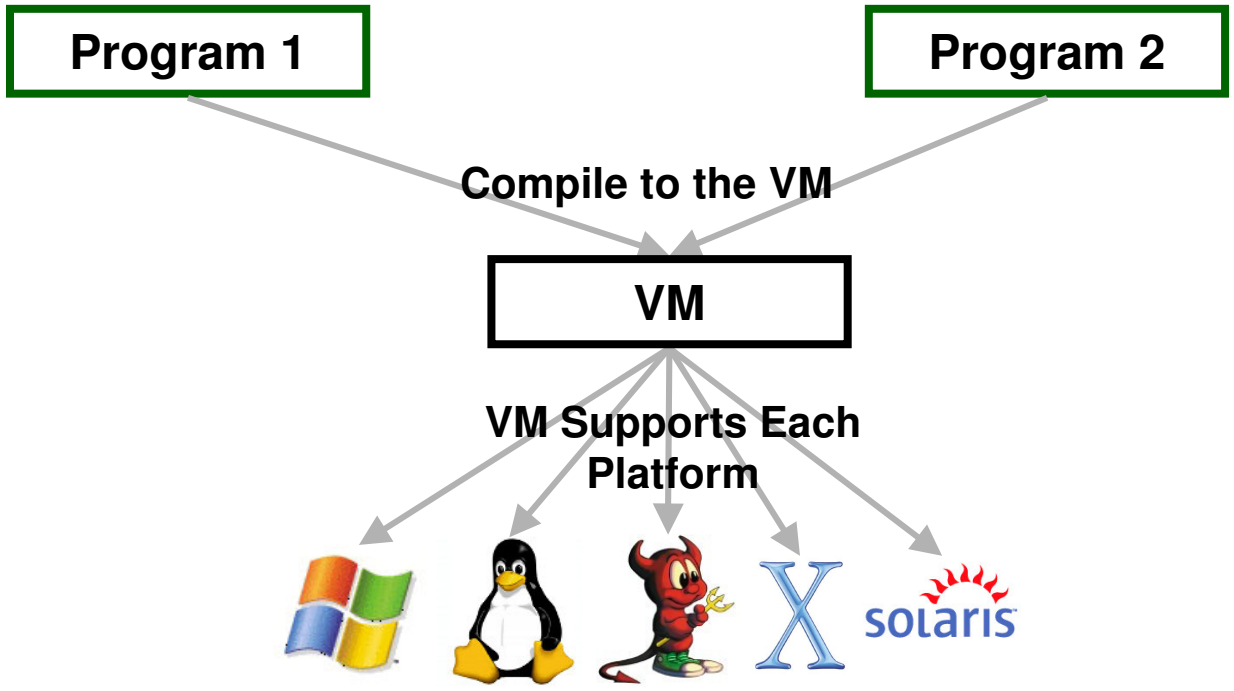


Without a VM

Inside Virtual Machines

Why Virtual Machines?

1. Simplified software development and deployment.



With a VM

Inside Virtual Machines

Why Virtual Machines?

2. High level languages have a lot in common.
   - Strings, arrays, hashes, references, …
   - Subroutines, objects, namespaces, …
   - Closures and continuations
   - Memory management

   Can implement these just once in the VM.

Inside Virtual Machines

Why Virtual Machines?

3. High level language interoperability becomes easier.
   - A consistent way to call subroutines and methods.
   - A common representation of data types: strings, arrays, objects, etc.
   - Code in multiple languages essentially runs as a single program.

Inside Virtual Machines

Why Virtual Machines?

4. Can provide fine grained security and quota restrictions.
    - "This program can connect to server X, but can not access any local files."
5. Debugging and profiling more easily supported.
6. Possibility of dynamic optimizations by exploiting what is known at runtime but not be known at compile time.

Inside Virtual Machines

A Few Well Known VMs

- The JVM (Java Virtual Machine)
- .Net CLR (Common Languages Runtime)
- Parrot
- Many things you might not call VMs…
  - For example, the Perl 5, or Python, or Ruby interpreter could in many ways be considered a VM; they are just closely tied to the language.

Stack and register architectures

Inside Virtual Machines

Stack and register machines

Most virtual machines, including .NET and JVM, are implemented as stack machines.

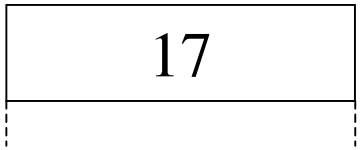
```
push 17

push 25

add
```

Inside Virtual Machines

Stack and register machines

Many virtual machines, including .NET and JVM, are implemented as stack machines.

push 17



push 25

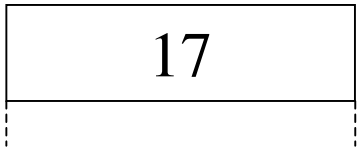
add

Inside Virtual Machines

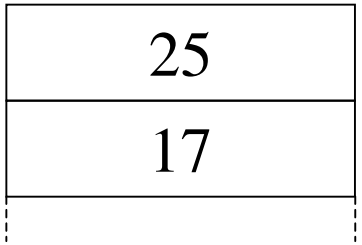
Stack and register machines

Many virtual machines, including .NET and JVM, are implemented as stack machines.

push 17



push 25



add

Inside Virtual Machines

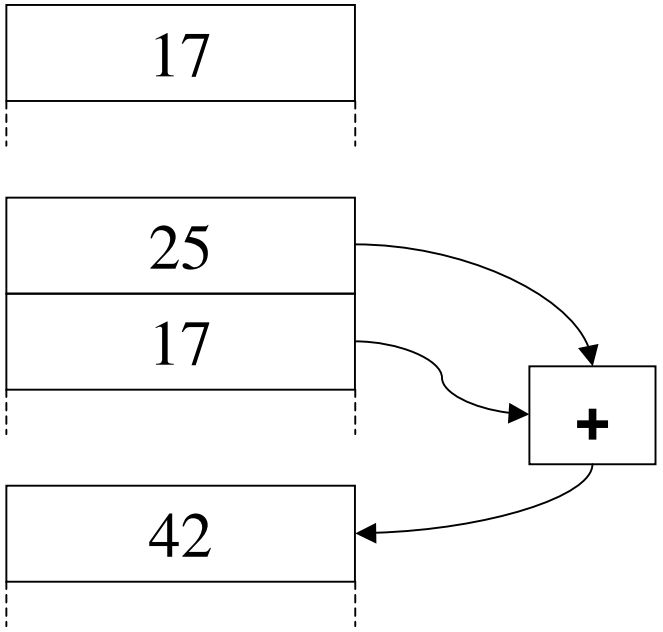
Stack and register machines

Many virtual machines, including .NET and JVM, are implemented as stack machines.

push 17

push 25

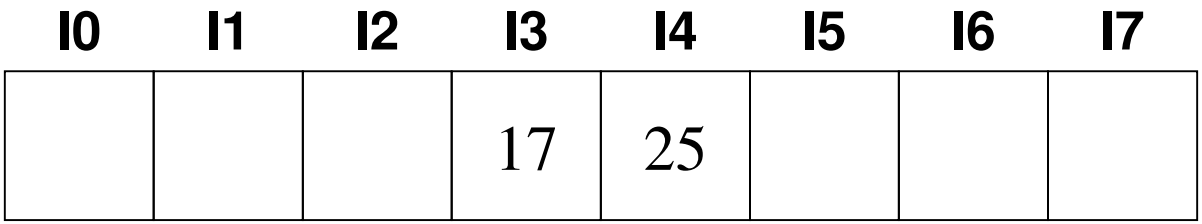
add



Inside Virtual Machines

Stack and register machines

Other virtual machines, such as Parrot, use registers. A register is a numbered storage location for holding working data.

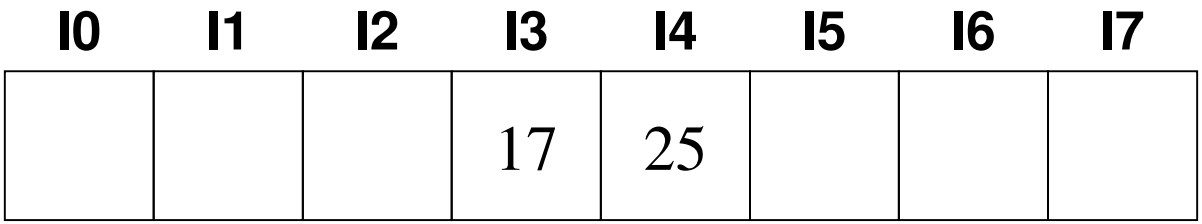


Inside Virtual Machines

Stack and register machines

The add instruction in Parrot adds the values stored in two registers and stores the result in a third.

add I1, I3, I4

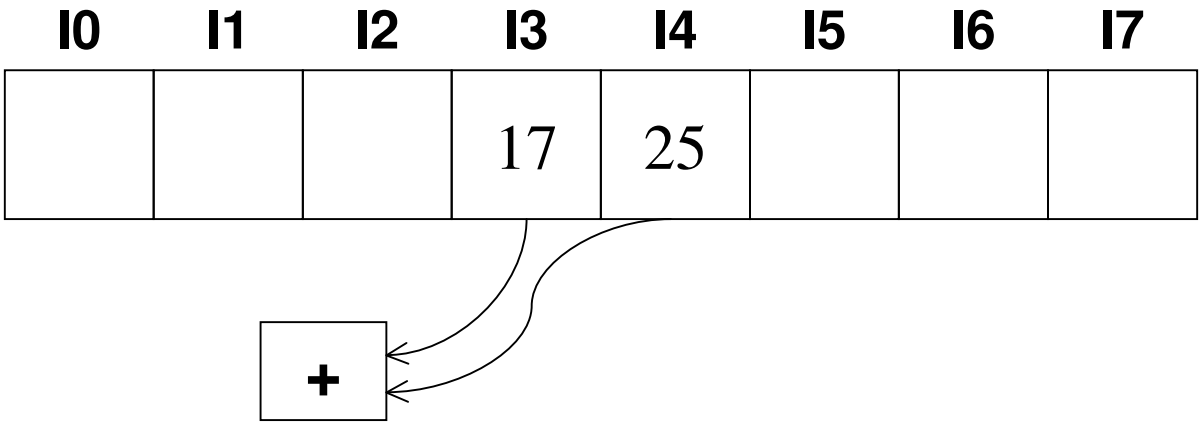


Inside Virtual Machines

Stack and register machines

The add instruction in Parrot adds the values stored in two registers and stores the result in a third.

add I1, I3, I4

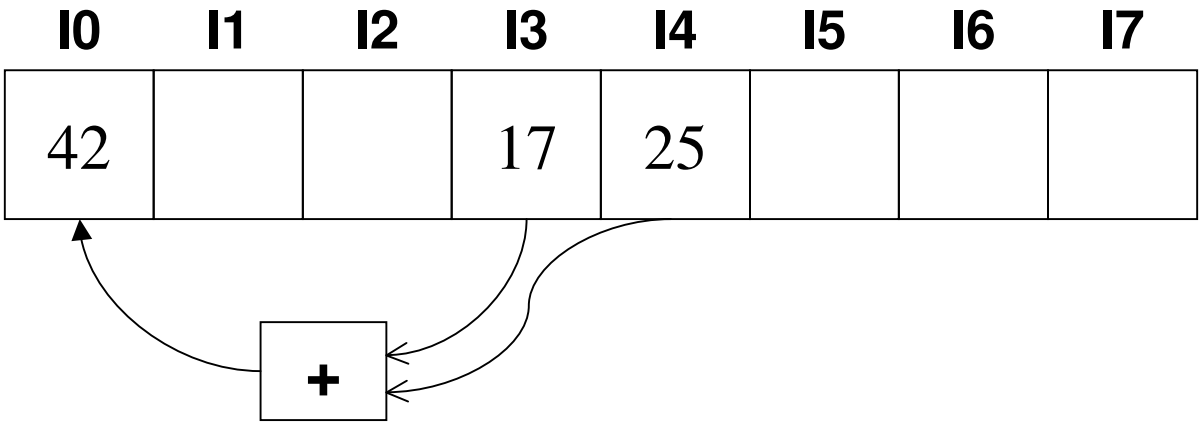


Inside Virtual Machines

Stack and register machines

The add instruction in Parrot adds the values stored in two registers and stores the result in a third.

add I0, I3, I4



Inside Virtual Machines

Register machine advantages

- What could be expressed in one register instruction took at least three stack instructions.
- When interpreting code (rather than JITing – more later), there is overhead for mapping each virtual instructions to a real one at runtime, so less instructions is better.

Running virtual machine code

Inside Virtual Machines

Running Virtual Machine Code

- There are a number of ways to execute code in the instruction set of the virtual machine on real hardware.
- Generally, the most portable solution (that works on most platforms) will be the slowest…
- …and the fastest ones will be the least portable.

Inside Virtual Machines

The "function per instruction" approach

- Have one C function per instruction.
- Build a big array of pointers to those functions; array index = instruction code.
- Execute instructions by looking up the function appropriate in the table then calling it.
- Completely portable, but performance hit due to making a function call per instruction.

Inside Virtual Machines

The "switch" approach

- A huge "switch" statement with one case for each instruction.
- After executing an instruction, the program counter is increment and we jump back to the top of the switch block again (using goto).
- Performance depends heavily on the code the compiler generates for switch blocks, but no per-op function call overhead is a bonus.
- Also completely portable.

Inside Virtual Machines

The "computed" goto approach

- GCC allows goto to jump to a memory address computed at runtime rather than a named label like most other compilers!
- Write C code for each instruction in a single function, prefix it with a label and build a table of label addresses.
- After executing each instruction, look up the address of the C code for the next instruction using the table and goto that address.

Inside Virtual Machines

The "computed" goto approach

- Computed goto performs better than the previous two approaches, worse than JIT.
- However, it only works on a small number of compilers, so not very portable.
- Code that uses computed goto interacts nastily with the C compiler's optimizer – basically the optimizer can't do much with it.
- Tends to mean that the computed goto core takes a lot of time and memory to compile.

Inside Virtual Machines

What is a JIT compiler?

- Just In Time means that a chunk of bytecode is compiled when it is needed.
- Compilation involves translating Parrot bytecode into machine code understood by the hardware CPU.
- High performance – can execute some Parrot instructions with one CPU instruction.
- Not at all portable – custom implementation needed for each type of CPU.

Inside Virtual Machines

How does JIT work?

- For each CPU, write a set of macros that describe how to generate native code for the VM instructions.
  - Do not need to write these for every instruction; can fall back on calling the C function function that implements it.
- A Configure script determines the CPU type and selects the appropriate JIT compiler to build if one is available.

Inside Virtual Machines

How does JIT work?

- A chunk of memory is allocated and marked executable if the OS requires this.
- For each instruction in the chunk of bytecode that is to be translated:
  - If a JIT macro was written for the instruction, use that to emit native code.
  - Otherwise, insert native code to call the C function implementing that method, as an interpreter would.

Memory Management

Inside Virtual Machines

Memory Management

- During their execution, programs allocate memory for storing working data in.
- Often this memory is only used for a short amount of time.
- There is only a finite amount of memory available to use, so programs need to free up memory that is no longer being used.
- Traditionally programs did this themselves, e.g. through malloc() and free() in C.

Inside Virtual Machines

What is GC (Garbage Collection) and why?

- Garbage collection systems automate the freeing of memory when it is no longer in use.
- The programmer is no longer responsible for freeing memory meaning:
  - No memory leaks.
  - No chance of accidentally freeing things that are still in use.
  - Faster development.

Inside Virtual Machines

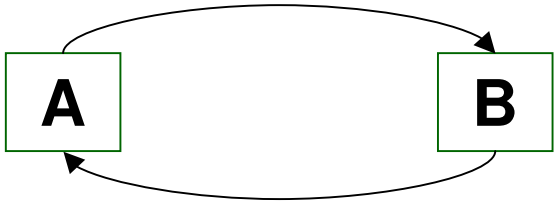
An "Easy" Solution: Reference Counting

- Just one approach to garbage collection, used in Perl 5 and many other interpreters.
- Every object has a reference count – a value that keeps track of the number of variables and other objects that refer to that object.
- When the reference count reaches zero, there is no way the object could be accessed, so it is no longer in use, therefore it can be freed.

Inside Virtual Machines

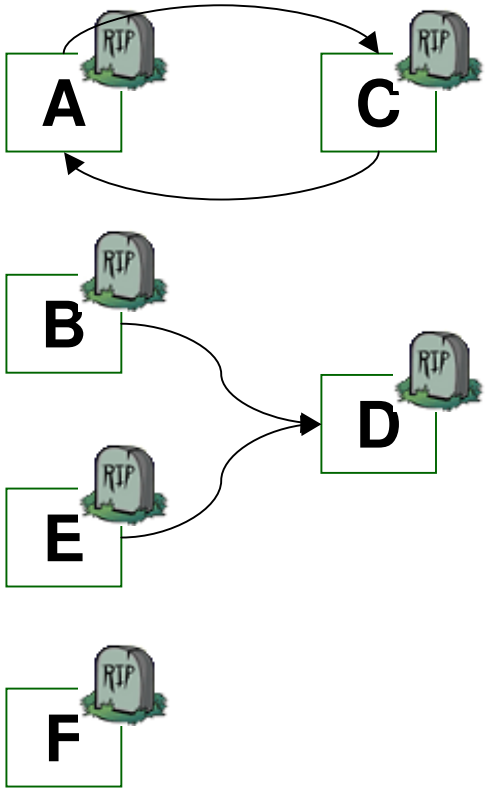
Reference Counting Not Really Easy

- Very easy to forget to increment or decrement the reference count as needed.
- VM code littered with reference count manipulation.
- Circular data structures never get freed as their reference count never reaches zero.



Reachability Based GC

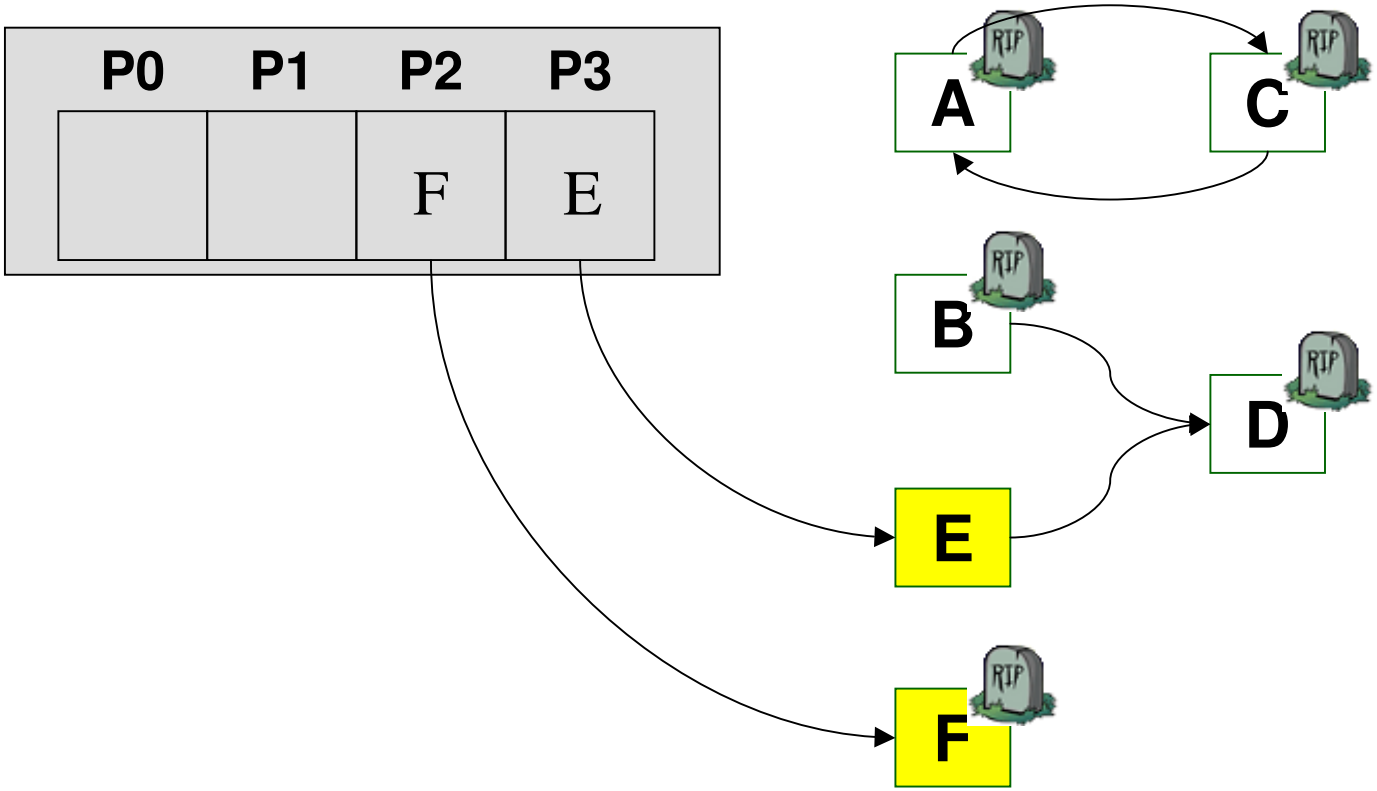
- Initially consider all objects dead (that is, unreachable).



Inside Virtual Machines

Reachability Based GC

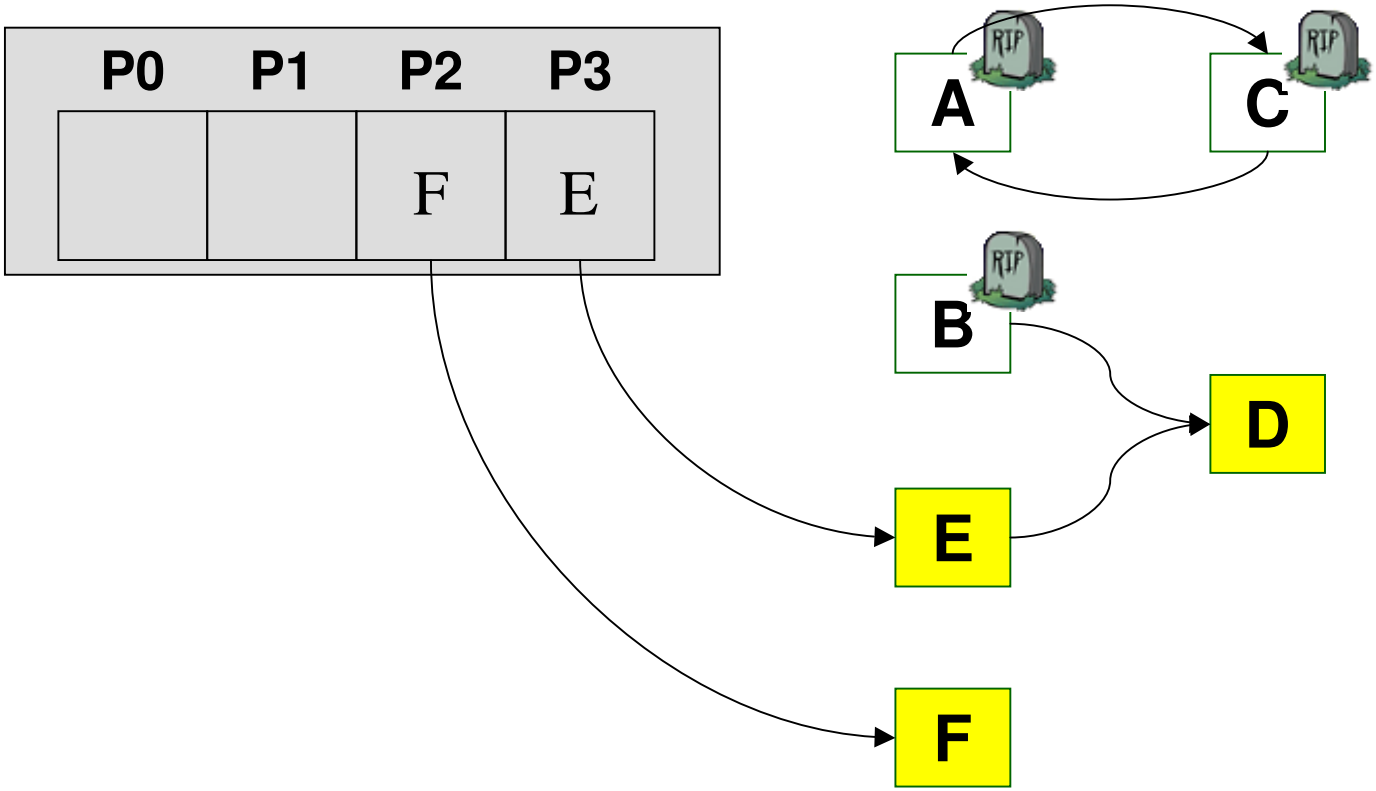
- Mark any objects that are referenced by registers or on the stack as live.



Inside Virtual Machines

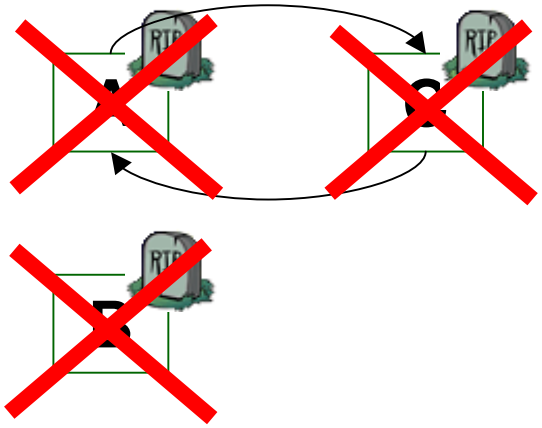
Reachability Based GC

Transitively mark objects referenced by live objects as alive.



Reachability Based GC

- Objects that were not marked alive can thus have the memory associated with them freed.



Developing Virtual Machines

Inside Virtual Machines

Regression Testing

- No two teams developing a VM are the same, but they all use regression testing.
- Each time a feature is added to the VM or a bug is found, write some test code that tests the feature or produces the bug.
- Tests can all be run automatically and their output checked.
- Breakage or bugs that re-surface will be spotted quickly.

Inside Virtual Machines

Regression Testing

- Can also write tests before features are implemented to ensure they work as expected when implemented.
  - Test Driven Development (TDD)
- Also useful for ensuring that multiple implementations of the same VM produce the same results.
  - Good for Harmony project, implementing open source JVM.

Inside Virtual Machines

Build Tools

- Build tools are often used to avoid writing a lot of boring and repetitive code by hand.
- For example, Parrot implements many different run-cores (function per instruction, switch, computed goto).
- The code for each instruction is only written once and the function headers, switch block or goto code is generated automatically.

Inside Virtual Machines

Platform Awareness

- It's important to try and write code that will run on platforms with…
  - Different compilers
  - Different byte order and word order
  - Different system APIs
  - A character encoding other than ASCII
- Some platforms are really weird.

Inside Virtual Machines

Getting Involved

- The first step is to download the source, compile and play with a VM…
  - http://www.parrotcode.org/
  - http://incubator.apache.org/harmony/
- Read the docs, play around, find bugs!
- Report ‘em, or write a patch – both are helpful.
- Who knows where it may lead…

The End

Any questions?