Java

Google

JavaOne℠

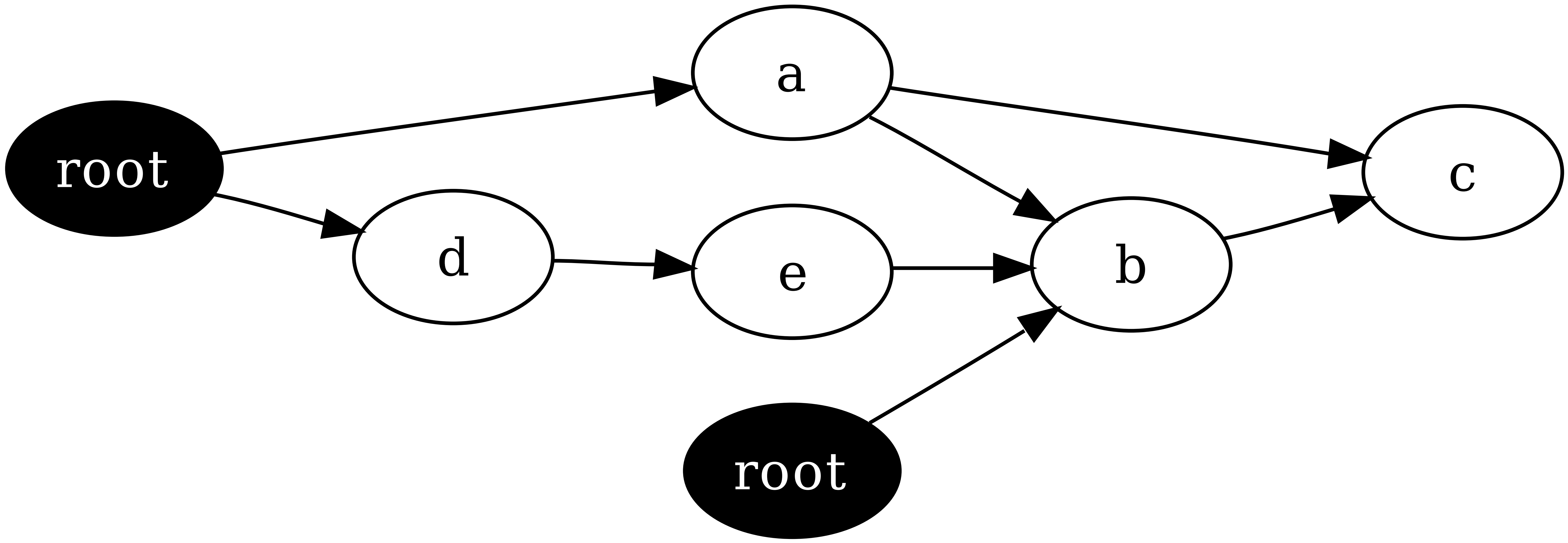
The Ghost in the Virtual Machine
A Reference to References

Bob Lee
Google Inc.

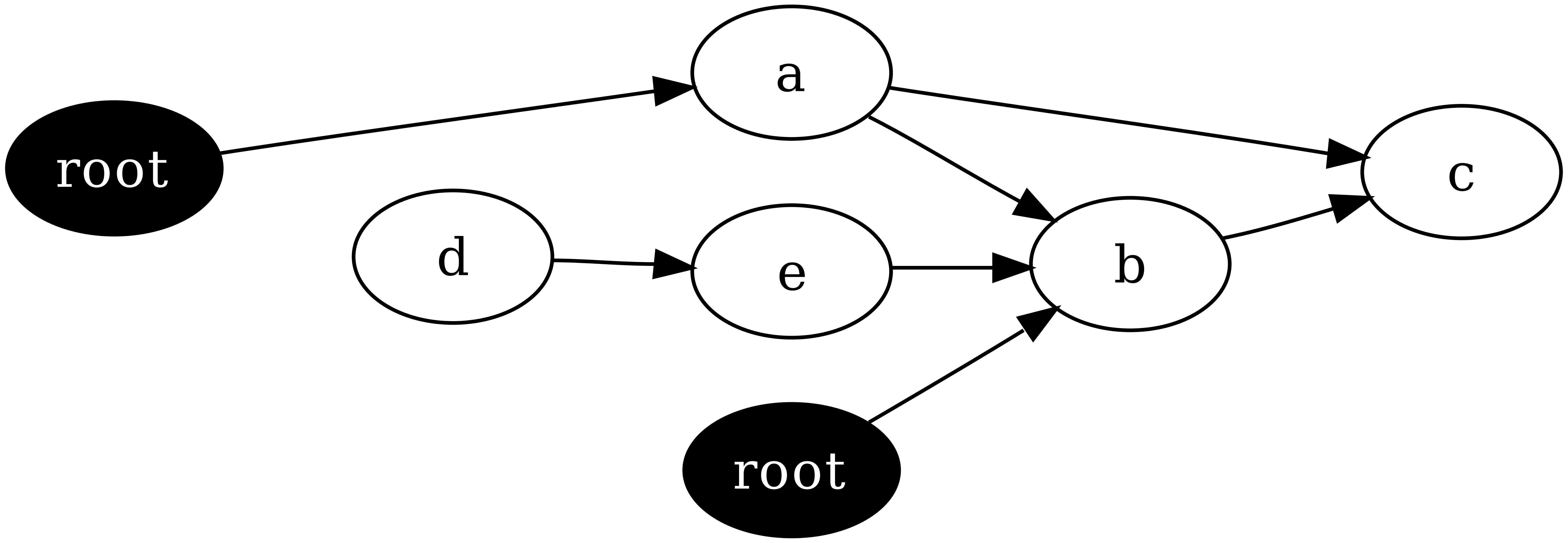
Goals

> Take the mystery out of garbage collection.
> Perform manual cleanup the right way.
> Become honorary VM sanitation engineers.

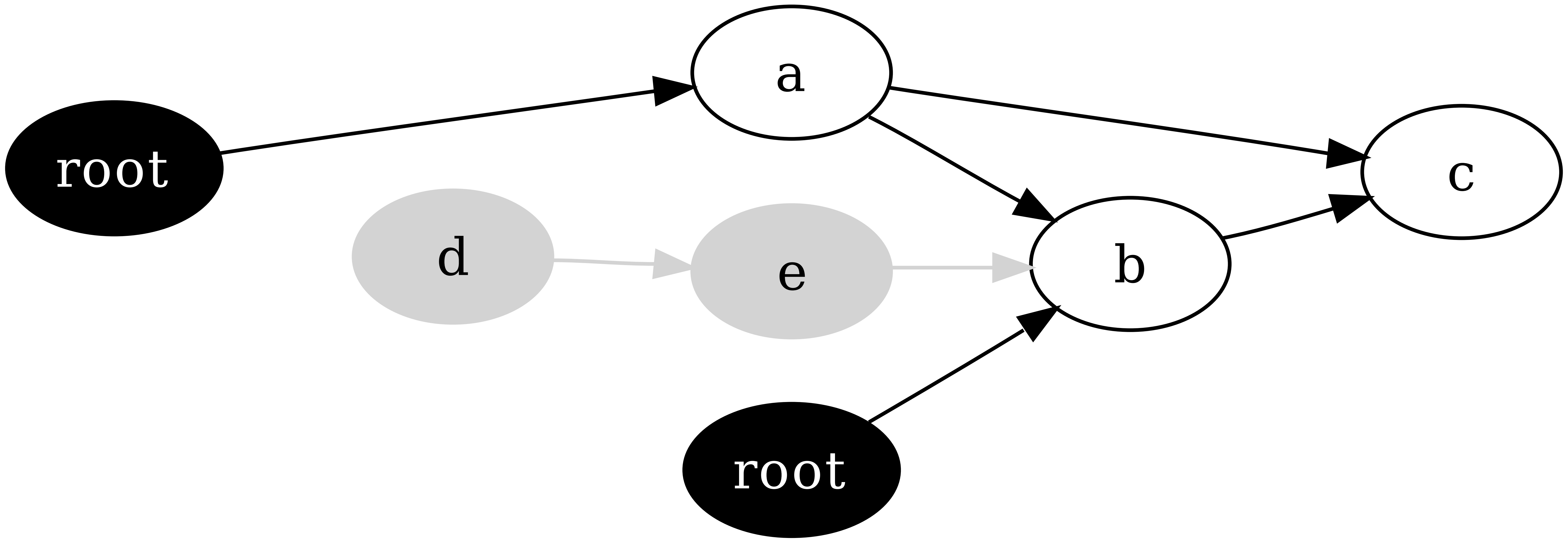
How does garbage collection work?



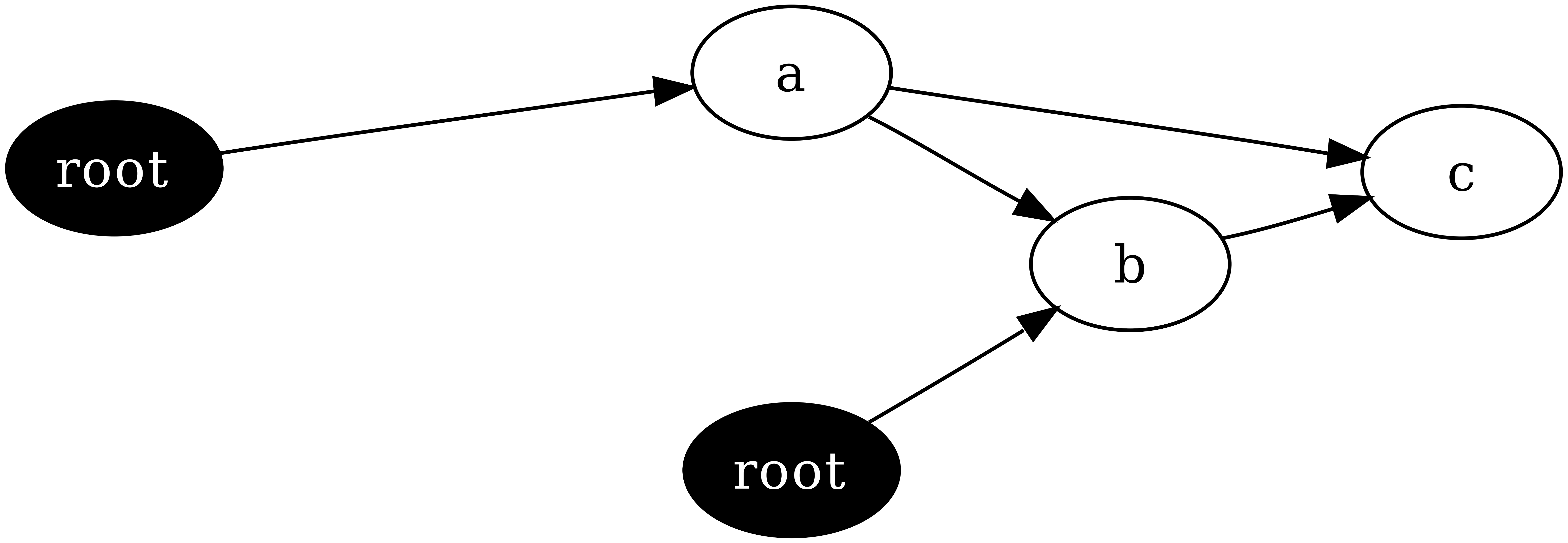
If the reference to D goes away...



We can no longer reach D or E.



So the collector reclaims them.



The GC can't do everything.

> Some things require manual cleanup.
  - Listeners
  - File descriptors
  - Native memory
  - External state (`IdentityHashMap`)

> Tools at your disposal:
  - `finally`
  - Overriding `Object.finalize()`
  - Reference queues

What is a finalizer?

Extend `NativeResource` and make it safe.

```
public class SafeNativeResource extends NativeResource {
  private boolean finalized;

  @Override public synchronized void write(byte[] data) {
    if (!finalized) super.write(data);
    else /* do nothing? */;
  }

  @Override protected synchronized void finalize() {
    finalized = true;
    super.finalize();
  }
}
```

An external resource

```
public class NativeResource {
  public NativeResource() { init(); }

  /** Allocates native memory. */
  private native void init();

  /** Writes to native memory. */
  public native void write(byte[] data);

  /** Frees native memory. */
  @Override protected native void finalize();
}
```

Let's play War!

`SegfaultFactory` can cause a segfault if its finalizer executes after `NativeResource`'s.

```
public class SegfaultFactory {
  private final NativeResource nr;
  public SegfaultFactory(NativeResource nr) {
    this.nr = nr;
  }

  @Override protected void finalize() {
    // 50/50 chance of failure
    nr.write("I'm taking the VM with me!".getBytes());
  }
}
```

Use protection.

Extend `NativeResource` and make it safe.

```
public class SafeNativeResource extends NativeResource {
  private boolean finalized;

  @Override public synchronized void write(byte[] data) {
    if (!finalized) super.write(data);
    else /* do nothing? */;
  }

  @Override protected synchronized void finalize() {
    finalized = true;
    super.finalize();
  }
}
```

Reachability

> An object is *reachable* if a live thread can access it.

> Examples of heap roots:

- System classes (which have static fields)
- Thread stacks
- In-flight exceptions
- JNI global references
- The finalizer queue
- The interned String pool
- etc. (VM-dependent)

Dante's Heap - The Levels of Reachability

> Strong
> Soft
> Weak
> Finalizer
> Phantom, JNI weak
> Unreachable

Dante's Heap - The Levels of Reachability

- **Strong**
- Soft
- Weak
- Finalizer
- Phantom, JNI weak
- Unreachable

Dante's Heap - The Levels of Reachability

> Strong
> **Soft**
> Weak
> Finalizer
> Phantom, JNI weak
> Unreachable

Dante's Heap - The Levels of Reachability

- Strong
- Soft
- **Weak**
- Finalizer
- Phantom, JNI weak
- Unreachable

Dante's Heap - The Levels of Reachability

> Strong
> Soft
> Weak
> **Finalizer**
> Phantom, JNI weak
> Unreachable

Dante's Heap - The Levels of Reachability

> Strong
> Soft
> Weak
> Finalizer
> **Phantom, JNI weak**
> Unreachable

Dante's Heap - The Levels of Reachability

> Strong

> Soft

> Weak

> Finalizer

> Phantom, JNI weak

> **Unreachable**

Two options for freeing native resources

- Use a finalizer.
  - You must defend against subsequent use!
- Or use a phantom reference.

Weak references aren't for caching!

> Many collectors will reclaim weak refs immediately.

> Use soft reference for caching, as intended:

“Virtual machine implementations are encouraged to bias against clearing recently-created or recently-used soft references.”

- The `SoftReference` documentation