ARC versus GC

Die Umstellung des Merlin-Projekts

CocoaHeads Aachen 23.2.2012

Das Merlin Projekt

- 9 Frameworks
- 5 Programme
- 18 Xcode-Projekte,
- ca. 15.000 Dateien und Ordner

Das Merlin Projekt

Mit Leopard umgestellt auf Garbage Collection

- Spart viel Nebenarbeit: Konzentration auf Kernkompetenz
- Bessere Qualität
- „On the long run, the higher abstraction always wins.“
- iOS?

GC für iOS?

- Erwartung, dass GC auch für iOS kommen muss
- iOS 4: noch nicht
- iOS 5: wir müssen auf iOS 6 warten
- Am 6. Juni 2011 hat das Warten ein Ende mit Schrecken: kein GC für iOS, ever.

GC für Mac?

Is GC (Garbage Collection) deprecated on the Mac?

Aus den Mountain Lion Release Notes:

GC remains an option for development in Mac OS X v10.7. You are strongly encouraged to consider ARC for new development. For existing codebases (both manual reference counting and GC), you are encouraged to "test the waters." This is, however, a non-zero amount of work and you should weigh that effort with your other priorities.

[illegible] für ARC.

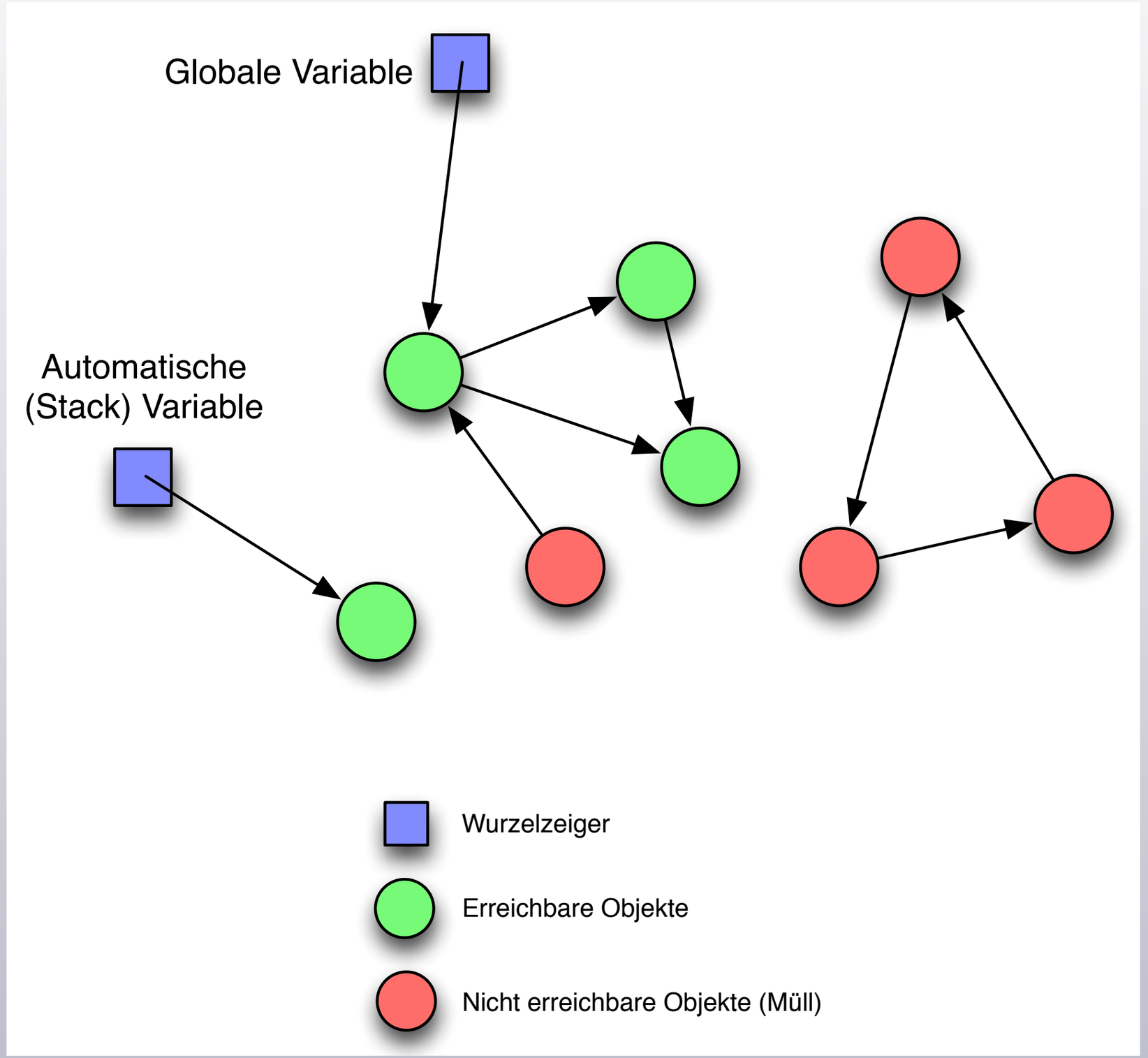
Wir trauen uns nicht, GC sofort sterben zu lassen. Aber glaubt nicht, wir würden noch Zeit reinstecken und Bugs fixen.

Resultat:

Umstellung des gesamten Merlin-Projekts auf ARC im Herbst 2011

CocoaHeads Aachen 23.2.2012

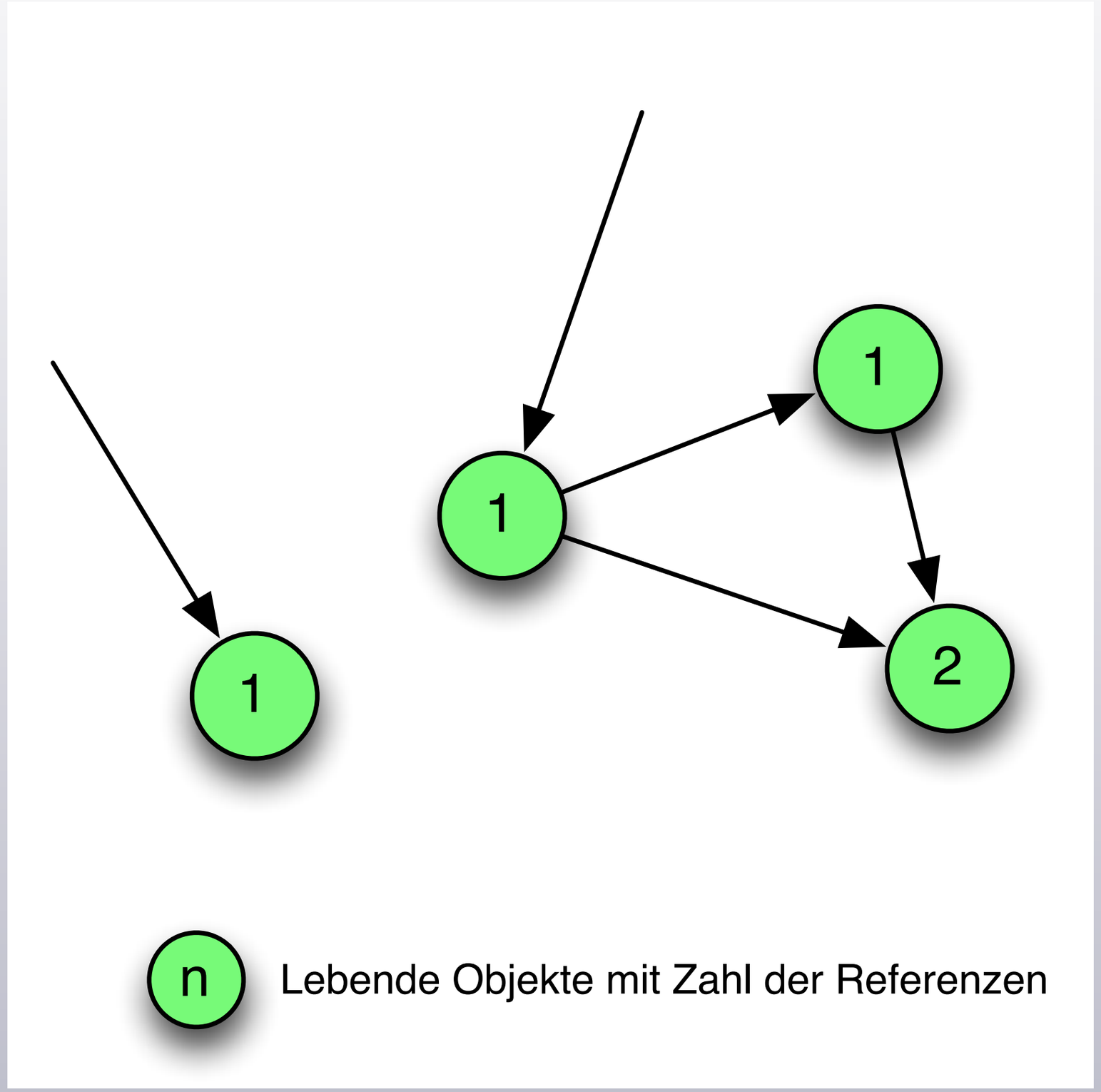
Garbage Collection



Garbage Collection

- Vollautomatisch
- Bewährt in modernen Sprachen wie Java, C#, Ruby, Javascript, etc.
- Standard für aktuelle Softwareentwicklung

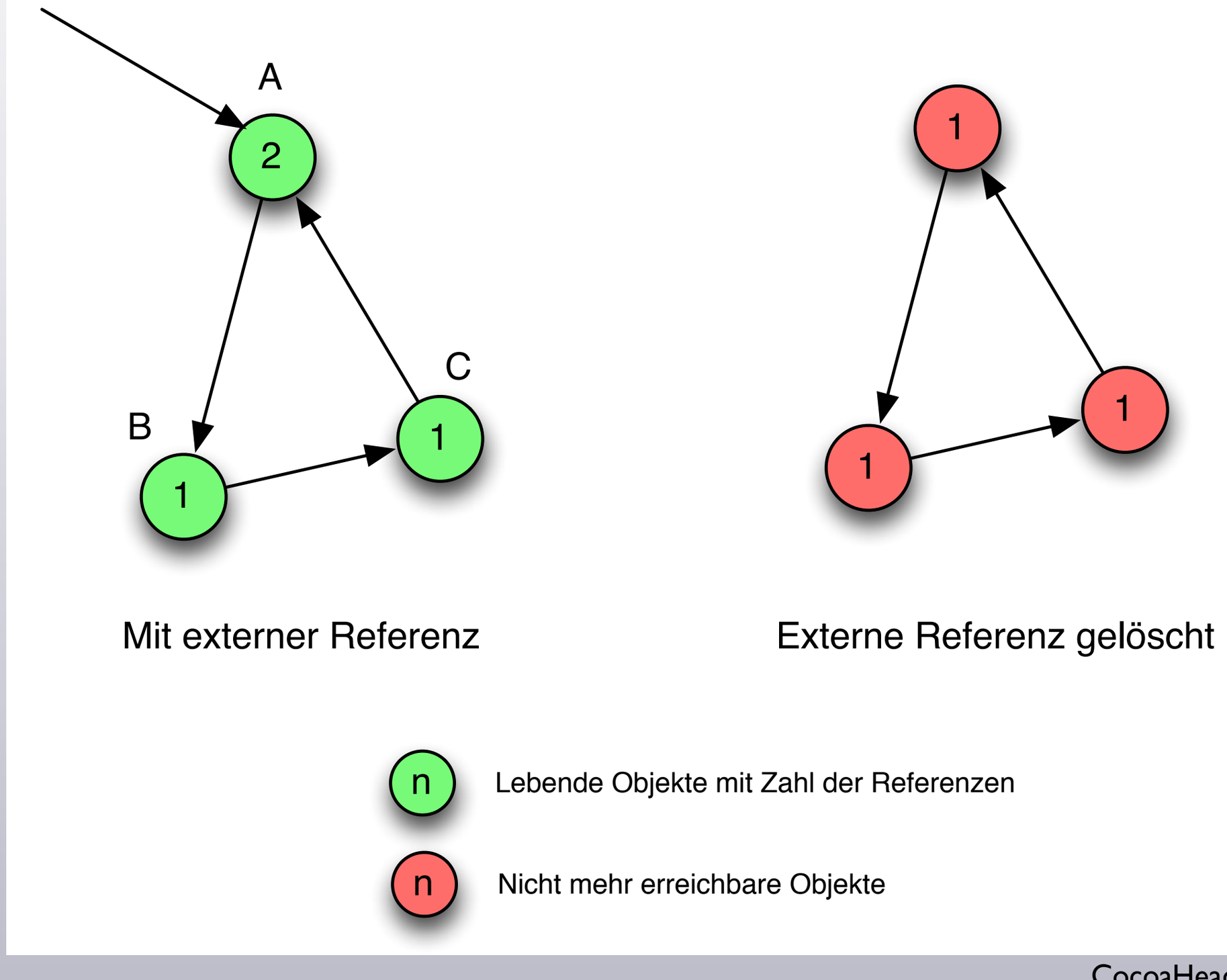
Reference Counting



Reference Counting

- Manuell mit -retain und -release
- Autorelease-Pool für Quasi-GC
- Underretain: Crash oder Sicherheitslücke
- Overretain: Speicherleck
- Verlangt nach unmenschlich perfekter Buchhaltung
- Prinzipielles Problem: zirkuläre Referenzen

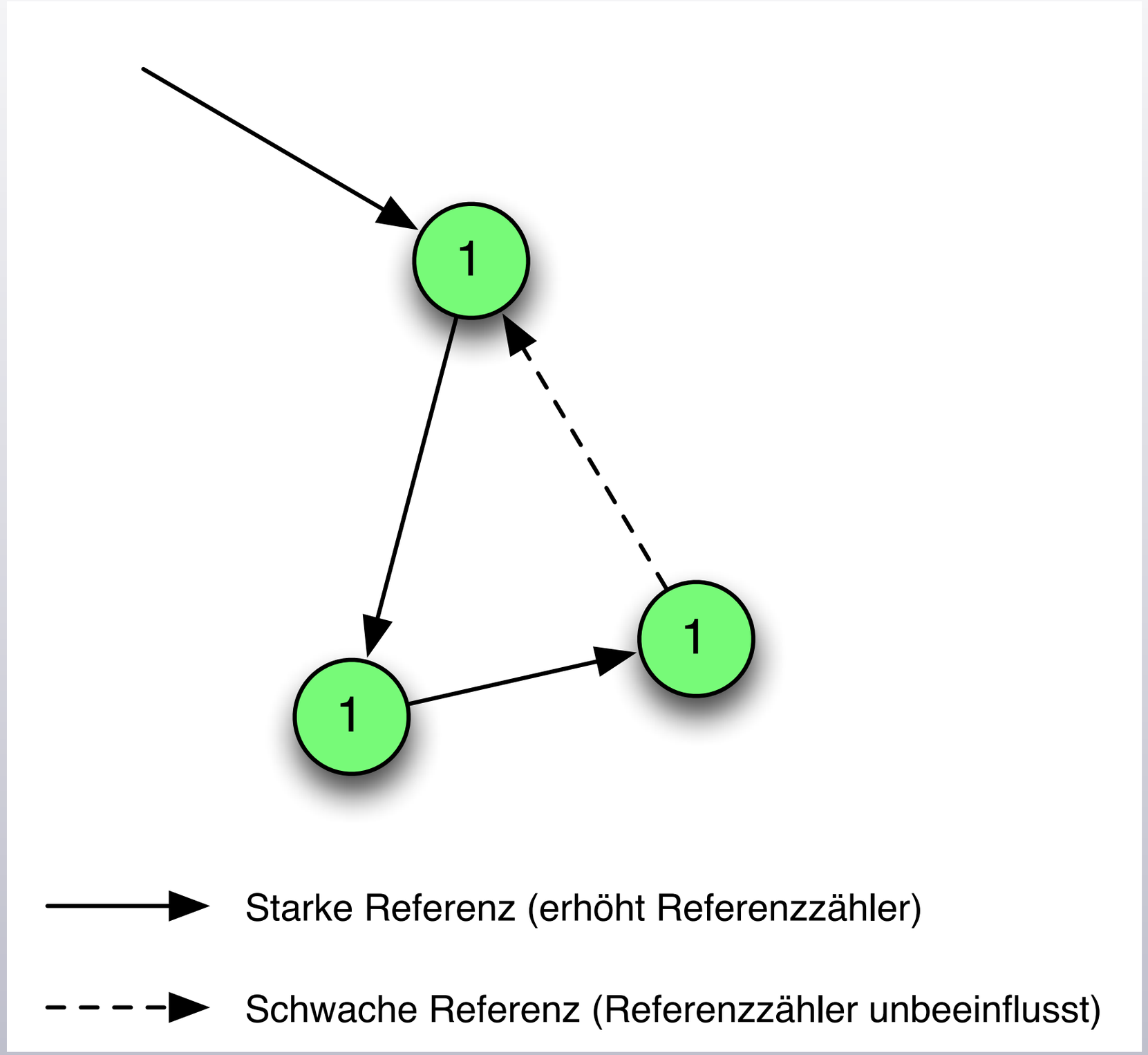
Zirkuläre Referenzen



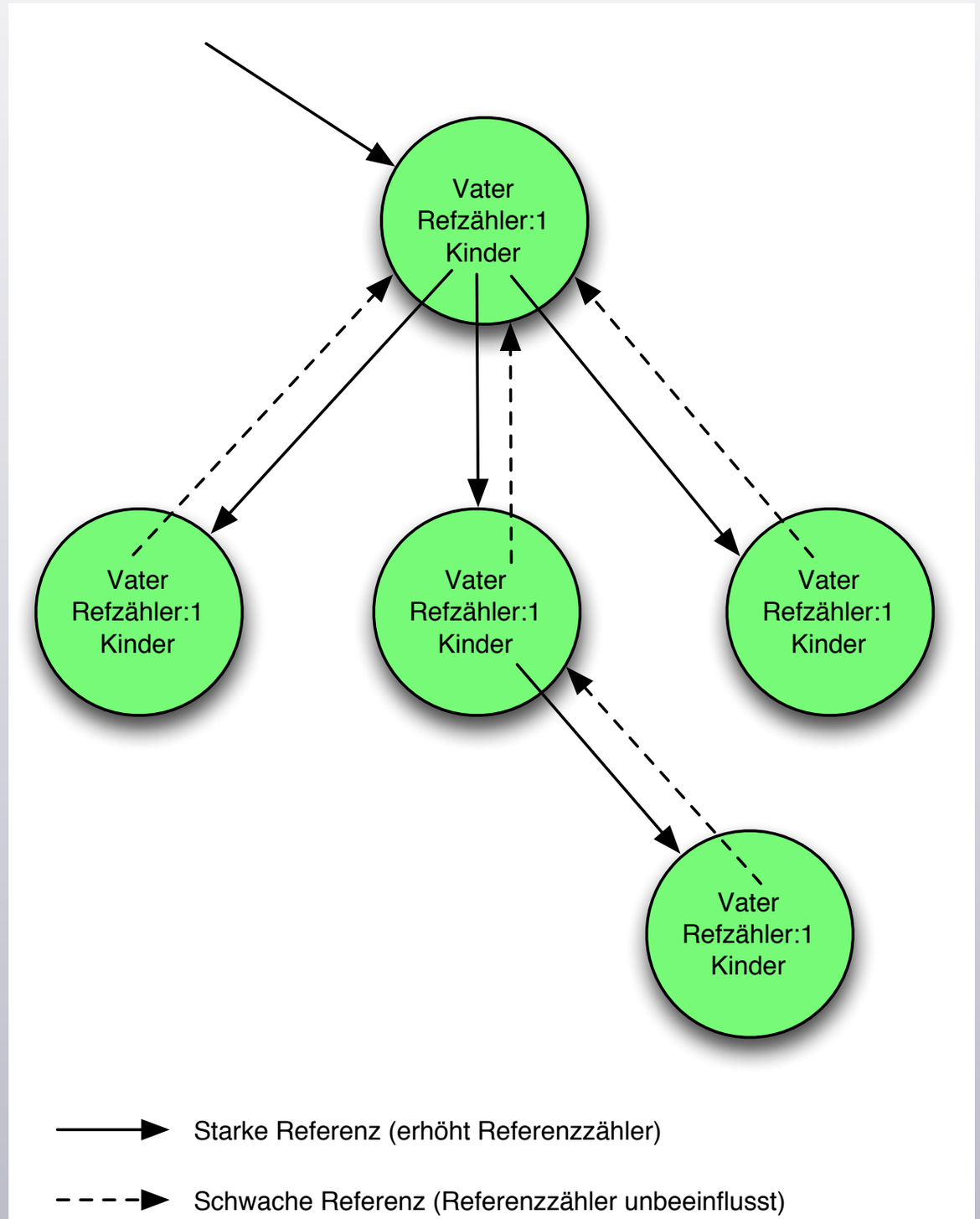
Zirkuläre Referenzen

- Aufbrechen durch „schwache Referenzen“: halten referenziertes Objekt nicht am Leben.
- Manuelle schwache Referenzen sind gefährlich

Schwache Referenz



Baum mit schwachen Referenzen



Schwache Referenzen

- Gut für gerichtete Strukturen (Delegate, Baum)
- Aber: Richtung manchmal unklar (Cocoa Text)

- Nicht geeignet für allgemeine Objektgraphen (CoreData) ➜ zweiter Referenzzähler
- `-tearDown`-Muster für Untersysteme
- Suche nach Zyklen bleibt mühsam

Nachteile von GC

- Collector stoppt das ganze Programm
- Allokierte Objekte kosten ständigen Scan-Aufwand
- Scans brauchen CPU-Zyklen und belasten die Batterie
- Speicher-Hochwassermarke liegt höher
- GC bringt nicht-Reproduzierbarkeit in den Programmlauf

ARC

- Compiler generiert retain, release & Co
  ➜ Buchhaltungsproblem ist gelöst
- „Zeroing weak references“ in der Runtime (ab Lion und iOS 5)
- Zyklen-Problem bleibt, eher schlimmer

ARC Lifetime-Qualifier

```
__strong
__weak
__unsafe_unretained
__autoreleased
```

Die ersten drei ohne __ auch für Properties.

ARC Bridge-Casts

Casts in und aus der ARC-Welt:

`__bridge`

`__bridge_retained` erhöht retain-count

`__bridge_transfer` verringert retain-count

```
CFStringRef cfString = …

[(__bridge NSString*)cfString doSomething]
```

`dealloc` unter ARC

- Oft leer, da ivars automatisch released werden
- Aufruf von `[super dealloc]` automatisch

ARC Autorelease-Pool

NSAutoreleasePool wird ersetzt durch:

```
@autoreleasepool {
    …code…
}
```

Migration nach ARC

- Xcode-Werkzeug für Manuelles R/R ➜ ARC
  - Entfernt `retain`, `release`, `autorelease`
  - Entfernt leere `dealloc`-Methoden
  - Fügt `__bridge` casts ein
- Werkzeug für GC ➜ ARC ab Xcode 4.3
  - Weniger wichtig

Migration von Merlin

- Zwei bis drei Wochen
- Größter Teil der Zeit für Lecksuche

Migrationsschritte

- Einige Tausend Property-Deklarationen mit `strong`, `copy` oder `weak` verzieren
- Alle Property-Deklarationen auf `nonatomic` setzen (`atomic`-Default ist ein GC-Artefakt)
- Compiler hilft beim Rest, hauptsächlich `__bridge`-Casts
- Erster Start ➜ Exception wegen illegaler weak reference

Das Drama mit weak

Instanzen einiger Klassen können nicht weak referenziert werden:

`NSATSTypesetter`, `NSColorSpace`, `NSFont`, `NSFontManager`, `NSFontPanel`, `NSImage`, `NSMenuView`, `NSParagraphStyle`, `NSSimpleHorizontalTypesetter`, `NSTableCellView`, `NSTextView`, `NSViewController`, `NSWindow`, and `NSWindowController`.

In addition, in OS X no classes in the AV Foundation framework support weak references.

Das Drama mit weak

- Delegates sollten unter ARC weak gehalten werden
- Geht nicht, falls der Delegate zu einer der verbotenen Klassen gehören kann ➜ bleibt `unsafe_unretained` (aka `assign`)
- Eine delegierende Klasse muss wissen, von welcher Klasse ihre Delegates sein können ➜ stellt das Muster auf den Kopf

➜ Wir haben einen `PWViewController` gebaut

Suche nach Zyklen

- Instruments hat dafür das Leaks-Instrument
- Mit grafischer Anzeige von Zyklen

Demo

Unit-Tests und Instruments

- Keine direkte Unterstützung in Xcode
- Breakpoint am Teststart setzen
- Instruments getrennt starten und an den Testprozess attachen
- Für jeden Durchlauf wiederholen

Garbage Collection müsste man haben, dann wäre das Leben schön!

Demo

Selbstgebauter Mini-Leakcheck für zentrale Objekte

- Buchführung über die Instanzen in -init und -dealloc
- Check am Ende von Unit-Tests auf nicht freigegebene Instanzen
- Kleiner Trick: Autorelease-Pool um `-setUp`, Test und `-tearDown` muss geschlossen sein: Override `-invokeTest`

Selbstgebauter Mini-Leakcheck für zentrale Objekte

```
@implementation PWTestCase

- (void) invokeTest
{
    @autoreleasepool {
        [super invokeTest];
    }

    [self afterAutoreleaseDrain];
}

- (void) afterAutoreleaseDrain
{
    // for sub class overrides
}

@end
```

Zyklen durch Blocks

Unter GC:

```
self.blockForLaterUse = ^{ [self doSomething]; };
```

Unter R/R:

```
__weak MyClass* weakSelf = self;
self.blockForLaterUse = ^{
    MyClass* strongSelf = weakSelf;
    [strongSelf doSomething];
};
```

Zyklen durch Blocks

Wo ist hier der Zyklus?

```
__block int i = 0;
self.blockForLaterUse = ^{
    NSAssert (i < 5, nil);
    NSBeep();
};
```

In Blöcken nur `NSCAssert` verwenden!

Ein Apple-Programmierer bei der WWDC 2011:

„ARC is a giant leak-producing machine“

Fazit

- ARC funktioniert
- Zyklen-Problem ist schwerwiegend
- GC ➜ ARC kaum vermeidbar
- Manuelles R/R ➜ ARC ist ein No-Brainer