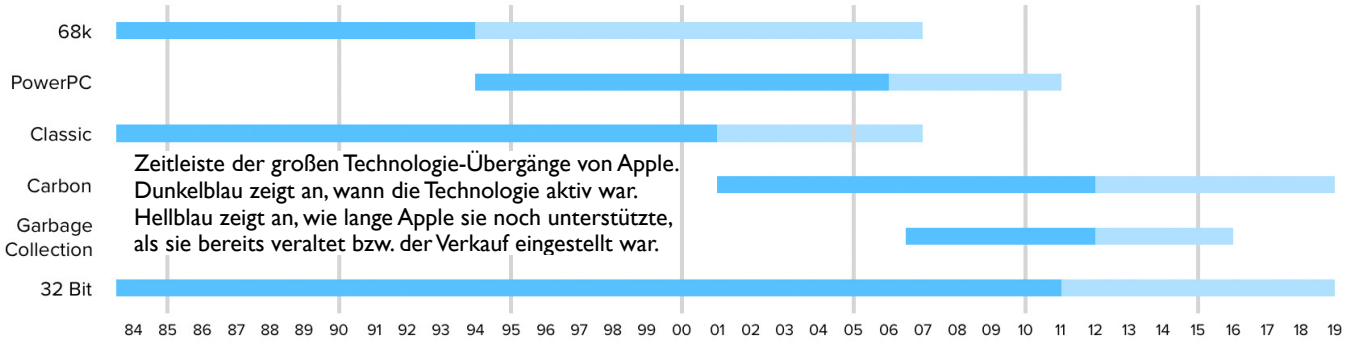


Apples Technologie-Übergänge

von Martin Pilkington (pilky.me), Übersetzung Kurt J. Meyer

Die Computerindustrie bewegt sich oft sehr schnell, und neue Technologien kommen und gehen mit schwindelerregender Geschwindigkeit. Einige glückliche Technologien sind gut genug durchdacht, um jahrzehntelang in Gebrauch zu bleiben, aber letztendlich gelten sie alle irgendwann als veraltet und werden durch neuere Technologien ersetzt.

Das Problem ist: Sobald man neue Technologien einführt, sei es Hard- oder Software, werden Entwickler damit beginnen, sie in ihren Apps zu verwenden, und Menschen und Unternehmen werden sich auf diese Apps verlassen. Daher ist es am Ende der Nutzungsdauer einer Technologie gar nicht einfach, sie etwas Anderes zu ersetzen. Sie könnten die Technologie einfach existieren lassen, aber auch das ist keine leichte Wahl, da sie Ressourcen für die Wartung benötigt und Sie u.U. daran hindern kann, an anderer Stelle Verbesserungen vorzunehmen.

Die richtige Balance zwischen Abwärtskompatibilität und Vorwärtsdynamik zu finden, ist unglaublich schwierig. So ist Microsoft dafür bekannt, sie auf die Spitze zu treiben und alte (manchmal auch uralte) Software so lange wie möglich zu unterstützen. Apple verfolgt einen anderen Ansatz, wechselt viel häufiger die Technologien und bricht damit die Abwärtskompatibilität. Das jüngste Beispiel dafür ist der Wegfall der 32-Bit-Anwendungen in macOS 10.15, der zu einer erheblichen Kontroverse geführt hat.

Die Kontroverse ist größtenteils dadurch begründet, dass viele Menschen das Wie oder Warum dieser Übergänge nicht verstehen und warum Apple letztendlich die alte Technologie fallen lässt. Also dachte ich, es sei von Nutzen, Apples Historie der Technologie-Übergänge zu erforschen und zu versuchen, einige der Gründe für diesen neuesten Übergang zu erklären, so dass ihn jeder verstehen kann.

Apples Übergänge in der Vergangenheit

Apples Übergänge können weitgehend in zwei Gruppen eingeteilt werden: CPU-Übergänge und API-Übergänge.

CPU-ÜBERGÄNGE

CPUs (Central Processing Units) sind das Herzstück eines jeden Computers. Im Grunde genommen nehmen CPUs eine Liste von Anweisungen und durchlaufen sie. Diese Anweisungen können das Abrufen von Daten aus dem Speicher Ihres Computers, das Addieren zweier Zahlen, das Vergleichen von zwei Werten und vieles mehr beinhalten. Die Anweisungen, die eine CPU ausführen kann, bezeichnet man zusammenfassend als ihren Befehlssatz. Wenn Sie eine Software für einen bestimmten Befehlssatz erstellen, dann läuft Ihre Software theoretisch auf jeder CPU, die diesen Befehlssatz verwendet, unabhängig davon, wer ihn erstellt hat. Umgekehrt bedeutet dies, dass die für einen Befehlssatz erstellte Software nicht auf einer CPU mit einem anderen Befehlssatz läuft. Im Laufe seiner Geschichte hat Apple die Befehlssätze mehrmals gewechselt.

68k auf PowerPC (1994)

Der erste Wechsel für den Mac war 1994 die Umstellung von der Motorolas 68k-Prozessorserie auf PowerPC-Prozessoren. In den frühen 90er Jahren begann der Mac, hinter anderen Computern, insbesondere Windows-PCs mit ihren Intel-Prozessoren, zurückzubleiben. Apple hatte auch Probleme mit Motorola. Deren neue 88k-Serie von Prozessoren wurde nicht gut aufgenommen und es gab Lieferprobleme mit der 68k-Serie.

Apple beschloss, sich anderswo umzusehen und arbeitete schließlich mit IBM zusammen, um eine Version ihrer POWER-Prozessoren zu entwickeln, die für Desktop- und Laptop-Computer geeignet war. Sie brachten auch Motorola ins Boot und bildeten die so genannte AIM-Allianz. Gemeinsam entwickelten sie den ersten PowerPC-Prozessor.

Da der Mac bereits über eine große Sammlung vorhandener Software für die 68k-Serie verfügte, wandte sich Apple die Emulationstechnologie an, um sicherzustellen, dass diese Software auf den neuen PowerPC-Macs laufen würde. Emulationssoftware simuliert effektiv einige physikalische Hardware per Software. In der Regel führt dies zu einem Performanceverlust, da Sie die vorhandene Hardware nicht mehr effektiv nutzen können, aber es stellt zumindest sicher, dass die Software weiterhin läuft.

Diese Emulationsschicht funktionierte unglaublich gut und ermöglichte es Apple, weiterhin Software zu unterstützen, die für 68k geschrieben wurde, bis 2007, als Mac OS X 10.5 die Unterstützung für die Classic-Umgebung (auf die wir später noch eingehen) einstellte.

PowerPC zu Intel (2006)

Als der PowerPC zum ersten Mal auf den Markt kam, war es eine sehr leistungsstarke CPU, die die Konkurrenz von Intel oft ausstach. Dies währte bis in die 90er Jahre, aber um die Jahrtausendwende stieß Apple auf weitere Probleme. Der PowerPC G4 wurde von Motorola produziert, aber seine Leistung war stagnierend und er kämpfte sich darum, über 1,7 GHz hinauszukommen. Es war um diese Zeit herum, als Apple anfing, sich die Intel-Prozessoren anzusehen, die sie regelmäßig schlugen.

Der PowerPC bekam eine kurze Gnadenfrist, als IBM den PowerPC G5 vorstellte, einen 64-Bit-Chip mit einem großen Leistungsschub über dem G4. Dies ermöglichte es Apple, den beliebten PowerMac G5 zu produzieren und die Leistungskrone wieder zu übernehmen. Leider tauchten einige Jahre später wieder ähnliche Probleme auf. IBM war nicht in der Lage, den G5 auf 3GHz zu bringen, was Intel und AMD regelmäßig taten. Noch schlimmer ist, dass Stromverbrauch und Heizleistung des G5 es unmöglich machten, ihn in Laptops einzubauen. So biss Apple 2005 in den sauren Apfel und kündigte an, es werde seine Produktpalette komplett auf Intel umstellen.

Ähnlich wie beim Übergang vom 68k benötigte Apple einen Weg, damit PowerPC-Software auf ihren neuen Intel-Macs ausgeführt werden konnte. Diesmal lizenzierten sie eine Technologie namens QuickTransit, die CPU-Befehle dynamisch von einem Befehlssatz in den anderen übersetzte und ihn zum Aufbau ihrer Rosetta-Funktion verwendete. Dadurch konnte die PowerPC-Software weiterlaufen, bis diese Funktion 2011 mit der Veröffentlichung von Mac OS X 10.7 eingestellt wurde.

Intel zu ARM

Der Übergang von Intel zur ARM-Architektur ist interessant, da er bisher weitgehend nur Apple betrifft. Bei der ersten Entwicklung des iPhone hatte Apple die Wahl, was das Betriebssystem sein sollte. Einige wollten es auf der Software des iPods aufbauen, die bereits auf ARM lief. Die Fraktion, die sich durchsetzte, waren jedoch diejenigen, die Mac OS X auf ARM portieren wollten, und das bildete die Grundlage für das, was wir heute als iOS kennen.

Da es nicht notwendig war, bestehende Software zu portieren und ältere Hardware zu unterstützen, konnte Apple diesen Übergang relativ einfach durchführen. Der Übergang ist jedoch möglicherweise noch nicht abgeschlossen. Da Apples A-Serie von CPUs immer leistungsfähiger wird, beginnen sie, mit Intels Angeboten zu konkurrieren und übertreffen sie am unteren Ende sowohl in Bezug auf die Leistung als auch auf die Energieeffizienz. Wenn sich dieser Trend fortsetzt, ist es sehr wahrscheinlich, dass ein Übergang zur ARM-Architektur (zumindest für Low-End-Macs) der nächste große CPU-Umstieg sein wird, den Apple durchführt.

Fat Binary-Dateien

Ein wichtiger Aspekt dieser Übergänge war die Verwendung von Fat Binaries durch Apple. Während Emulation und Übersetzung es alten Softwareprodukten ermöglichen, auf neuer Hardware zu laufen, ohne dass der Entwickler etwas ändern muss, gibt es weiterhin viele Leute mit alter Hardware, die gern aktuelle Software verwenden möchten, auch wenn diese nativ neue Hardware erfordert.

Die grundlegendste Möglichkeit, dies zu tun, wäre für einen Entwickler, zwei Versionen einer App zu liefern, eine, die auf der alten Hardware läuft, und eine, die auf der neuen läuft. Leider erfordert dies, dass Entwickler diese unterschiedlichen Versionen verwalten und dass die Benutzer sicherstellen müssen, dass sie die richtige Version erhalten. Und wenn der Benutzer seine Hardware aktualisiert, muss er die gesamte Software neu installieren.

Fat Binaries bieten eine elegantere Möglichkeit. Sie beinhalten mehrere Versionen des ausführbaren Codes in einer einzigen App. So konnte beispielsweise ein einzelner App-Download nativen Code für PowerPC und Intel enthalten. Das bedeutet, dass genau die gleiche App auf einem PowerPC-Mac und einem Intel-Mac ausgeführt werden kann und auf beiden mit nativer Geschwindigkeit. Apple hat diese Technologie bei allen CPU-Übergängen angewendet, einschließlich des 32- bis 64-Bit-Übergangs (wie wir später sehen werden).

API-ÜBERGÄNGE

APIs (Application Programming Interfaces) sind Codegruppen, die von Plattformanbietern bereitgestellt werden, um Entwicklern das Schreiben von Software zu ermöglichen. Sie handhaben ein breites Spektrum von Dingen: vom Lesen und Schreiben von Dateien über das Abrufen von Daten via Netzwerk bis hin zu mathematischen Funktionen, um Bilder und UI-Elemente auf dem Bildschirm zu zeichnen. Es ist üblich, dass sich diese APIs im Laufe der Zeit weiterentwickeln, wobei neue APIs hinzugefügt und alte für obsolet erklärt werden (was ein Hinweis darauf ist, dass Entwickler sie nicht mehr verwenden sollten und dass sie irgendwann verschwinden können).

Gelegentlich kann aber eine API auch vollständig ersetzt werden. Dies erfordert in der Regel ein Umschreiben großer Code-Passagen in Apps und kann sogar ein grundsätzliches Überdenken der Funktionsweise einer App nötig machen. Es gibt drei große Änderungen, die sich auf den Mac beziehen:

Classic zu OS X

Als Apple Mac OS X einführte, war dies die größte Veränderung der Software auf fast allen wichtigen Plattformen. Alles wurde von oben nach unten verändert, als Apple das 1996 erworbene NeXTStep-Betriebssystem mit Mac OS fusionierte. Diese Änderungen waren so umfangreich, dass so ziemlich jede vorhandene Mac-Software nicht ohne wesentliche Änderungen auf Mac OS X laufen konnte.

Um dieses Problem zu lösen, führte Apple die Classic-Umgebung auf Mac OS X ein, die Mac OS 9, die letzte Version des „Classic" Mac OS, virtualisierte. Virtualisierung ist der Emulation ähnlich, simuliert aber keine Hardware, die nicht auf Ihrem Computer vorhanden ist. Stattdessen segmentiert es Ihren Computer effektiv und ermöglicht es, mehrere Betriebssysteme gleichzeitig auszuführen. Die Betriebssysteme laufen beide nativ auf der Hardware, aber sie teilen sich CPU, RAM, Speicher etc.

Wenn Sie eine Classic-Anwendung unter Mac OS X starteten, wurde ein Fenster angezeigt, das das Booten von Mac OS 9 anzeigte. Sobald das fertig war, konnte die App starten. Sie konnten das Fenster wie bei jedem nativen Mac OS X-Fenster verschieben, obwohl es deutlich unter Mac OS 9 angezeigt wurde und sich auf die Funktionalität von Mac OS 9 beschränkte. Leider lief Mac OS 9 nicht auf Intel-CPUs, so dass es auf den ersten 2006 veröffentlichten Intel-Macs nicht mehr unterstützt wurde. Im Jahr 2007 wurde es mit der Veröffentlichung von Mac OS X 10.5 komplett eingestellt.

Carbon zu Cocoa

Als Apple NeXTStep und Mac OS mit Mac OS X fusionierte, fusionierten sie auch die Methoden zum Erstellen von Software. Dazu war es erforderlich, sowohl alten Mac-Anwendungen als auch alten NeXTStep-Anwendungen einen einfachen Weg zur Migration zu bieten. Das Ergebnis waren zwei konkurrierende APIs: Cocoa und Carbon.

Cocoa basiert auf den NeXTStep-APIs. Diese APIs verwendeten die Objective-C-Sprache und haben eine reiche Historie; sie wurden verwendet, um Software wie den ersten Webbrowser, Macromedia FreeHand und die Level-Editoren für Spiele wie Doom und Quake zu entwickeln. Diese APIs wurden größtenteils so übernommen, wie sie sind, obwohl einige Funktionen geändert wurden, um die eingeführten Mac OS-Aspekte des Betriebssystems wiederzugeben.um die eingeführten Mac OS-Aspekte des Betriebssystems wiederzugeben.

Auf der anderen Seite basierte Carbon auf den Mac Toolbox APIs von Mac OS, die die Sprache C verwendeten. Dies ermöglichte eine viel einfachere Portierung klassischer Mac-Software auf Mac OS X und wurde zur Portierung von Software wie Microsoft Office und Adobe Photoshop verwendet.

Ursprünglich schien es, dass die beiden Plattformen nebeneinander existieren sollten, aber im Laufe der Zeit benötigten immer mehr neue Betriebssystem-Features Cocoa. Der Beginn des Endes von Carbon begann mit der Einführung von 64-Bit-Anwendungen. Während Cocoa 64-Bit wurde, tat es Carbon nicht. Carbon wurde 2012 als veraltet erklärt und als Teil von macOS 10.15 entfernt.

Garbage Collection

Dieser letzte API-Umstellung ist eine kleine Eigenart für Apple. Normalerweise, wenn eine neue Technologie veröffentlicht wird, ist es die alte Technologie, die entfernt wird. Objective-C Garbage Collection dauerte weniger als ein Jahrzehnt, bevor sie entfernt wurde, während die Technologie, die sie ersetzen sollte, weiterlebt.

Garbage Collection (GC) ist eine Möglichkeit für Software, RAM zu verwalten. Traditionell, wenn eine Software etwas RAM verwenden wollte, musste sie einen Teil des RAM „zuweisen". Sobald es mit diesem RAM fertig war, musste es es dann „freigeben", um es für den Rest des Systems zur Verfügung zu stellen. Dies ist eine fehleranfällige Methode, die zu Abstürzen und zum Aufblähen des Arbeitsspeichers führen kann, wenn sie nicht perfekt ausgeführt wird. GC versucht, dies für Entwickler zu automatisieren, indem es den Speicher im Auge behält; es erkennt, wenn etwas Speicher nicht mehr benötigt wird, und bereinigt ihn regelmäßig.

Apple hat die Garbage Collection für Objective-C im Jahr 2007 mit Mac OS X 10.5 eingeführt. Dies war eine beachtliche Leistung, da Objective-C eine Erweiterung der Programmiersprache C ist, die sich für GC nicht gut eignet. Leider hat GC Nachteile. Durch die regelmäßige Bereinigung des Speichers kann es in Software, die sich außerhalb der Kontrolle des Entwicklers befindet, zum vorübergehenden Stottern kommen. Obwohl GC verhindert, dass der Speicher aufgrund von Fehlern außer Kontrolle gerät, benötigt es mehr Speicher zum Ausführen als manuell verwalteter Code. Und die Objective-C Garbage Collection unterstützte keine C-basierten APIs, so dass die oben genannten Speicherfehler immer noch sehr wahrscheinlich waren.

All dies stellte ein Problem für das iPhone mit seinen begrenzten Ressourcen dar; daher hat es die Garbage Collection nie nach iOS geschafft. Stattdessen führte Apple die automatische Referenzzählung (ARC) ein. Anstatt wie GC den Speicher zu verfolgen, während die Anwendung läuft, bestimmt ARC schon, wenn die Software kompiliert wird, wann ein Speicherbereich zugewiesen und wieder freigegeben werden muss. Das vermeidet den erhöhten Speicherverbrauch und das periodische Stottern von GC. Es brachte letztendlich das Beste aus beiden Welten hervor und ermöglichte zugleich die Unterstützung einiger Objective-C-APIs.

Leider bedeutete dies, dass der Garbage Collector obsolet war, so dass Apple ihn 2012 mit Mac OS X 10.8 für veraltet erklärte, nur 5 Jahre nach seiner Einführung. Er wurde 2016 mit macOS 10.12 endgültig entfernt. Zufälligerweise ist dies der erste Übergang, bei dem eine von mir geschriebene Software unbrauchbar wurde, da ich die Entwicklung eingestellt hatte, bevor ich sie auf ARC umstellen konnte.

Als Apple beides auf einmal tat

Eine Sache, die Sie an diesen Übergängen bemerken werden, ist, dass Apple immer einen Schritt nach dem anderen macht. Sie haben entweder die CPU oder die APIs gewechselt, aber nie beides auf einmal. Es gab jedoch ein Vorkommen bei Apple, wo es beides zugleich getan hat, und das war im Grunde der erste große Übergang, den sie vollzogen haben: der Übergang vom Apple II zum Macintosh.

Der Apple II war einer der ersten erfolgreichen Personalcomputer. Die 1977 erstmals entwickelte Apple II Linie wurde bis 1993, neun Jahre nach Einführung des Macintosh, weiterentwickelt und verkauft. Alle Apple IIs verwenden eine 6500er CPU, die ursprünglich von MOS Technology entwickelt wurde. Es lief auch Apple DOS und später ProDOS als Betriebssystem. Diese Unterschiede verhinderten, dass Apple II - Software auf dem Mac ausgeführt wurde, was teilweise zum dauerhaften Erfolg des Apple II beitrug.

Schließlich stellte Apple den Apple II ein, als die Verkaufszahlen deutlich unter denen des Mac lagen. Apple hat jedoch die Lebensdauer der Apple II Software in einer — zumindest für Apple — einzigartigen Weise verlängert. Sie führten die Apple IIe Erweiterungskarte für den Mac ein. Dadurch brachte die Apple II-Hardware effektiv auf eine Erweiterungskarte, und erlaubte es in Verbindung mit Emulationssoftware, Apple II-Software auf dem Mac auszuführen, zumindest noch bis 1996, als die Karte nicht länger angeboten wurde.

64 Bits

Der Übergang, über den ich am meisten sprechen möchte, ist aber der Übergang von 32-Bit zu 64-Bit. Die letzte Phase dieses Übergangs hat schließlich diesen Beitrag ausgelöst. Dies ist grundsätzlich ein weiterer CPU-Übergang, unterscheidet sich aber von den Befehlssatzübergängen. Anstatt zu bestimmen, welche Anweisungen ausgeführt werden können, beeinflusst diese Änderung die Größe des Datenstücks, an dem die CPU in einem einzigen Vorgang arbeiten kann. So kann beispielsweise eine 32-Bit-CPU mit bis zu 32 Nullen und Einsen in einer einzigen Anweisung arbeiten. Wenn es an mehr Daten arbeiten möchte, muss es diese in separate 32-Bit-Blöcke aufteilen.

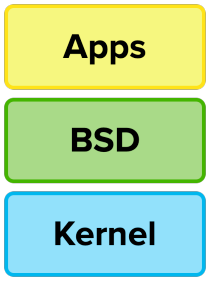
Dies wirkt sich vor allem auch darauf aus, wie viel RAM Ihr System haben kann. RAM ist praktisch ein Raster von Bytes (jedes Byte enthält 8 Nullen und Einsen), auf die in einer Reihenfolge zugegriffen werden kann (daher der Name Random Access Memory). Um auf eines dieser Bytes zuzugreifen, benötigen Sie eine Adresse, die nur eine Zahl ist. Entscheidend ist, dass die Größe dieser Zahl durch die Bitgröße begrenzt ist, die der Prozessor verwenden kann. Auf einem 32-Bit-System ist diese Nummer 232 oder 4.294.967.296. So kann ein 32-Bit-Prozess auf bis zu 4,29 Milliarden Bytes RAM oder 4 GB zugreifen.

Als 32-Bit eingeführt wurde, war dies kein Problem, da schon ein paar MB RAM als das Reich der Supercomputer galten. Aber um die Jahrtausendwende näherten wir uns schnell der Grenze von 4 GB, so dass die Arbeit an der Einführung von 64-Bit auf PCs begann. Es mag schwer zu verstehen sein, wie groß eine 64-Bit-Zahl ist, da die maximale Anzahl, die sie aufnehmen kann, 264 oder 18,4 Trillionen beträgt. Um einzuschätzen, wie groß diese Zahl ist, stellen Sie sich vor, dass Sie bis zu 4,29 Milliarden zählen, der maximalen Zahl, die Sie in 32 Bits halten können, und sagen Sie 1 Zahl pro Sekunde, ohne zu schlafen, zu essen oder zu trinken. Um das zu tun, würden Sie 136 Jahre brauchen. Wenn Sie nun weiter mit der gleichen Rate bis 18,4 Trillionen zählen würden, würde es 584,9 Milliarden Jahre dauern, oder etwa 42 mal das Alter des Universums. Unnötig zu sagen, dass es ziemlich unwahrscheinlich ist, dass wir in absehbarer Zeit wieder auf dieses Problem mit RAM stoßen werden.

Apples 64-Bit Übergang

Apples erster 64-Bit-Computer war der PowerMac G5, der 2003 auf den Markt kam. Es wurden spezielle Builds von Mac OS X 10.2.7 und 10.2.8 für diese Maschinen veröffentlicht, aber es wurde keine Software unterstützt, die tatsächlich die 64-Bit-Fähigkeiten des Prozessors nutzt.

Der Übergang von Apple zu 64-Bit erfolgte schrittweise über mehrere Versionen von Mac OS X, zwischen 2003 und 2007. Dazu trug bei, dass die G5 (und später auch die 64-Bit-Intel-CPUs) immer noch 32-Bit-Code nativ mit wenig bis keinem Leistungsverlust ausführen können. Um zu verstehen, wie Apple diesen Übergang vorgenommen hat, müssen wir einen kurzen Abstecher darüber machen, wie Mac OS X aufgebaut ist. In einer stark vereinfachten Skala gibt es drei Schlüsselebenen von Mac OS: den Kernel, die BSD-Schicht und die App-Schicht.



- Der Kernel ist der Kern des Betriebssystems, der die Feinheiten von Dingen wie Speicherverwaltung und Zugriff auf Speicher, Netzwerk und Peripheriegeräte verwaltet.
- Die BSD-Schicht ist eine Reihe von UNIX-Tools und APIs, die keine Benutzeroberfläche erfordern. Wenn Sie das Terminal überhaupt verwenden, haben Sie Werkzeuge in dieser Ebene verwendet.
- Die App-Schicht ist der Ort, an dem sich die bereits erwähnten Cocoa- und Carbon-APIs befinden. Alle Anwendungen, die Sie ausführen, die eine Benutzeroberfläche haben, werden in dieser Schicht ausgeführt.

Apple hat sich entschieden, diese drei Schichten in aufeinander folgenden Betriebssystem-Versionen zu migrieren, was den Übergang zu 64-Bit sowohl für Entwickler als auch für Anwender erleichterte.

Mit Mac OS X 10.3 (veröffentlicht 2003) migrierte Apple den Kernel auf 64-Bit, zusammen mit einigen leistungsempfindlichen Mathematik-APIs. Dies bedeutete, dass ein 64-Bit-System mehr als 4 GB RAM verbrauchen konnte, und Entwickler konnten Software schreiben, die einige der 64-Bit-Fähigkeiten der CPU nutzte, wenn auch ein ziemlich begrenztes Set.

Mit Mac OS X 10.4 (veröffentlicht 2005) migrierte Apple die BSD-Schicht auf 64-Bit. Dies bedeutete, dass einzelne Prozesse auf dem Computer auf mehr als 4 GB RAM zugreifen konnten (während 10.3 dem Gesamtsystem erlaubte, auf mehr als 4 GB RAM zuzugreifen, war der einzelne Prozess immer noch auf jeweils 4 GB beschränkt). Entwickler konnten nun 64-Bit umfangreicher nutzen, da die BSD-Schicht viele APIs für Plattenzugriff, Netzwerk, Bildverarbeitung usw. enthält. Allerdings mussten sie diese separat für den UI-Teil ihrer App ausführen, was die Komplexität erhöhte.

Schließlich hat Apple mit Mac OS X 10.5 (veröffentlicht 2007) die Cocoa-APIs auf 64-Bit umgestellt. Dies bedeutete, dass Entwickler eine vollständige 64-Bit-App ohne Kompromisse erstellen konnten. Ähnlich wie beim Übergang zu Intel wurde dies mit Fat Binaries erreicht, was bedeutete, dass zwischen 2007 und 2011 viele Apps (und das Betriebssystem selbst) tatsächlich 4 Versionen enthielten, jeweils eine für 32-Bit PowerPC, 64-Bit PowerPC, 32-Bit Intel und 64-Bit Intel.

Eine Kombination aus diesem schrittweisen Ansatz und Fat Binaries bedeutete, dass die Migration für die Benutzer relativ nahtlos verlief. Es war für Entwickler etwas komplizierter, aber wohl viel einfacher als die vielen anderen Übergänge, die sie in den letzten zehn Jahren bewältigen mussten. Apple hat 2011 endlich den Support für 32-Bit-Macs eingestellt, 4 Jahre nach der Auslieferung der letzten 32-Bit-Hardware, obwohl der Support für die Ausführung von 32-Bit-Software bis 2019 weiter bestand.

Wie Microsoft zu 64-Bit überging

An dieser Stelle ist es sinnvoll, diesen Ansatz mit der Vorgehensweise von Microsoft beim 64-Bit-Übergang zu vergleichen. Microsoft hat 2005 die Windows XP Professional x64 Edition veröffentlicht. Im Gegensatz zu Mac OS X war dies eine andere Betriebssystemversion, bei der man Windows neu installieren musste.

Während viele 32-Bit-Anwendungen gut liefen (dank 64-Bit-CPUs, die 32-Bit-Software nativ ausführen), bot dies Microsoft eine seltene Gelegenheit, den Support für einige Softwareprodukte einzustellen. 64-Bit-Versionen von Windows erforderten neue Gerätetreiber, und die Unterstützung für ältere 16-Bit-Software wurde eingestellt. Sein kompletter Satz an 32-Bit-APIs wurde jedoch beibehalten.

Seitdem hat Microsoft 32- und 64-Bit-Editionen jeder Windows-Version veröffentlicht. Dies hat, zusammen mit den langlebigen 32-Bit-APIs, dazu geführt, dass viele Software nur 32-Bit geblieben ist, auch wenn der Großteil der Computer heute mit 64-Bit-Windows installiert ist. Diese Tatsache ist teilweise für die Probleme mit dem Wegfall von 32-Bit auf dem Mac verantwortlich. Der Intel-Switch erleichterte es Entwicklern, besonders Spiele-Entwicklern, ihre Software auf den Mac zu portieren. Aber da sich viele dieser Entwickler hauptsächlich auf Windows konzentrierten, behielten viele ihre Software in 32-Bit-Versionen.

Warum 32-Bit wegfallen lassen?

Das Wegfallen der Unterstützung für 32-Bit-Software hat nur teilweise mit 32-Bit selbst zu tun. Es überrascht nicht, dass die Auslieferung von 32- und 64-Bit-Versionen in einem Programmpaket mehr Speicherplatz beansprucht. Die 32-Bit-Systembibliotheken nahmen unter macOS 10.14 über 500 MB ein. Sie benötigen auch viel RAM.

Jede App, die auf Ihrem Computer läuft, muss auf eine Teilmenge der System-APIs zugreifen. Anstatt diese für jede App separat in den Speicher zu laden, lädt das Betriebssystem sie einmal und teilt sie mit allen Apps. Wenn Sie mehrere Anwendungen ausführen, spart dies enorme Mengen an RAM. Das Problem ist, dass die 32- und 64-Bit-Versionen dieser APIs separat geladen werden müssen. Zwar ist macOS klug genug, diese erst bei Bedarf in den Speicher zu laden, aber sie bleiben dort, nachdem sie geladen wurden. Das bedeutet, dass, wenn Sie eine einzelne 32-Bit-App starten, auch nur für eine Sekunde, 32-Bit-System-APIs einen Teil Ihres Speichers belegen, bis Sie das System neu starten.

iOS litt unter genau dem gleichen Problem (nicht überraschend angesichts der gemeinsamen Basis mit MacOS). Leider haben iOS-Geräte nicht die gleiche Speicher- oder RAM-Kapazität wie ein Mac, so dass dies ein viel kritischeres Problem war. Obwohl Apple erst 2013 das erste 64-Bit-iPhone mit dem 5S auf den Markt brachte, verzichtete man mit iOS 11 im Jahr 2017, knapp 4 Jahre später, ganz auf die Unterstützung für 32-Bit-Software.

Während diese Einsparung von Festplattenspeicher und RAM-Nutzung sicherlich auch dem Mac zugute kommt, gibt es unbestreitbar wichtigere Gründe für Apple, die 32-Bit-Unterstützung auf dem Mac wegfallen zu lassen. Sie haben eigentlich nicht viel mit 32-Bit selbst zu tun, sondern eher mit Entscheidungen, die 2007 getroffen wurden, als 64-Bit finalisiert wurden.

Die Objective-C Runtime

Objective-C ist die wichtigste Programmiersprache für Cocoa. Während Swift in den letzten Jahren durchgestartet ist, ist die überwiegende Mehrheit der CocoaAPIs immer noch Objective-C. Objective-C ist eine stark runtime-basierte Sprache. Eine Runtime ist ein Code, der die Umgebung, in der eine App läuft, einrichtet und steuert. Einige

Sprachen haben sehr kleine Runtimes, die nach dem Start der App wenig bewirken. Andere, wie Objective-C, interagieren mit der Runtime während der gesamten Laufzeit der App.

Als Apple mit Mac OS X 10.5 64-Bit einführte, führte man auch Objective-C 2.0 ein. Dazu gehörte auch eine neue und verbesserte Runtime, die dazu gedacht war, Probleme mit der alten Runtime zu beheben. Leider waren diese Korrekturen nicht mit bestehenden Apps kompatibel, so dass sie sich entschieden haben, diese Runtime nur in 64-Bit zur Verfügung zu stellen. Dies bedeutete jedoch, dass die nun veraltete Legacy-Runtime so lange noch beibehalten werden musste, wie 32-Bit-Anwendungen vorhanden waren.

Eines der Probleme, die diese neue Runtime behoben hat, ist ein ganz wichtiges: die zerbrechlichen *ivars*. Dies wird jetzt ein wenig Erklärung benötigen, aber hoffentlich werden Sie am Ende verstehen, wie der Wegfall der 32-Bit-Anwendungen und damit der veralteten Objective-C-Runtime Apple helfen wird, den Mac in Zukunft zu verbessern. Ich werde zunächst einige grundlegende Programmierkonzepte erklären, also zögern Sie nicht, die nächsten Absätze zu überspringen, wenn Sie bereits Programmierer sind.

Objective-C ist eine objektorientierte Programmiersprache. Das bedeutet, dass die grundlegenden Bausteine, die Entwickler verwenden, Objekte sind. Ein Objekt enthält eine Kombination von Daten und Funktionen, um auf diese Daten zu reagieren. Jedes Datenstück kann von Zahlen über Text bis hin zu anderen Objekten reichen. In Objective-C werden diese Daten als Instanzvariablen, kurz *ivars* genannt.

In der Legacy-Runtime werden *ivars* beim Kompilieren Ihrer Anwendung in einer Liste dargestellt. Um auf eine *ivar* zugreifen zu können, muss die Runtime den Offset vom Anfang des Objekts kennen. Im folgenden Beispiel sehen Sie, dass wir Namen, Alter und Eltern-ivars haben. Die Zahlen auf der linken Seite sind der Offset vom Anfang. Name ist um 0 versetzt, wie am Anfang der Liste. Alter ist um 8 versetzt, da die Textdaten darüber 8 Bytes zum Speichern benötigen. Die übergeordnete *ivar* wird durch weitere 4 (die Anzahl der Bytes, die zum Speichern einer Zahl benötigt werden) versetzt, wobei der Offset 12 beträgt:

```
Person
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
```

Das nächste Konzept in der objektorientierten Programmierung ist die Idee der Vererbung. Ein Objekt kann von einem anderen erben, indem es alle seine *ivars* und Funktionen übernimmt und weitere hinzufügt. Angenommen, Apple stellt uns das Person-Objekt als Teil seiner APIs zur Verfügung, aber wir wollen auch speichern, ob die Person Haustiere besitzt oder nicht. Wir können ein neues Objekt erstellen, das von Person erbt und erweitert wird. Es würde so aussehen:

```
HaustierbesitzerPerson
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
20 besitztFische: Boolean
21 besitztKatzen: Boolesch
22 besitztHunde: Boolean
```

Man kann sehen, dass die von uns hinzugefügten *ivars* am Ende angebracht wurden. Nun haben wir ein Problem, wenn Apple das Person-Objekt aktualisieren möchte, da dann diese Versätze aufgehoben sind. Lassen Sie uns sagen, dass Apple eine *ivar* hatKinder hinzufügt. In der Legacy-Runtime würde nun Folgendes passieren:

```
Person
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
20 hatKinder: Boolesch
HaustierbesitzerPerson
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
~~20 besitztFishe: Boolean~~
21 besitztKatzen: Boolesch
22 besitztHunde: Boolean
```

Plötzlich funktionieren unsere HaustierbesitzerPerson-Objekte nicht mehr, da sowohl die *ivar* hatKinder als auch die *ivar* besitztFisch den gleichen Offset haben. Die neue Objective-C-Runtime hat die Fähigkeit, die Offsets zu verschieben, wenn sie einen solchen Konflikt erkennt, was uns folgendes Resultat liefern würde:

```
Person
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
20 hatKinder: Boolesch
HaustierbesitzerPerson
0 Name: Text
8 Jahre alt: Nummer
12 Elternteile: Person
20 hatKinder: Boolesch
21 besitztFische: Boolean
22 besitztKatzen: Boolesch
23 besitztHunde: Boolean
```

Das Verhalten der Legacy-Runtime bedeutet effektiv, dass Apple seine bestehenden Objekte niemals mit neuen *ivars* aktualisieren kann, ohne bestehende Anwendungen zu zerstören. In Wirklichkeit haben sie Wege gefunden, dies zu umgehen, aber sie erweisen sich als sehr schwierig, was die Zeit, die Apple für neue Funktionen und Bugfixes aufwenden kann, reduziert. Tatsächlich hat die Schwierigkeit, die alte Runtime zu unterstützen, sehr wahrscheinlich dazu beigetragen, dass einige APIs auf iOS (das nur die neue Runtime verwendet) es nicht zurück auf den Mac geschafft haben.

Beachten Sie, dass es sich bei den hier verwendeten Zahlen und Datentypen nur um vereinfachte Beispiele handelt, nicht um tatsächliche Werte oder Typen. Für ein Beispiel mit Ist-Werten, die Sie in Objective-C finden würden, siehe diesen Beitrag von Greg Parker, der als Referenz für meinen Beitrag diente.

Andere Gründe

Während das *fragile ivars*-Problem ein Hauptgrund für den Wunsch ist, auf 32-Bit zu verzichten, steht es aber auch beispielhaft für ein größeres Problem, das Apple zu lösen hofft. Apple hat eine Menge Hinterlassenschaften in seinem Betriebssystem. Die Carbon-APIs sind seit vielen Jahren veraltet, da sie nie auf 64-Bit migriert wurden, aber für 32-Bit-Anwendungen müssen sie noch existieren. Einige dieser APIs gehen möglicherweise auf den ursprünglichen Mac zurück. Es gibt viele andere APIs, die mit 64-Bit eigentlich veraltet sind, in einer ähnlichen Situation mit vielen Workarounds in ihrem Code, um sie weiterhin funktionsfähig zu halten.

Das Problem mit diesen APIs besteht darin, dass immer weniger Entwickler sie verwenden und dass es immer weniger Leute gibt, die Fehler bemerken können, besonders mögliche Sicherheitsmängel. Es gibt auch weniger Ressourcen innerhalb von Apple, die in die Pflege dieser APIs investiert werden, um solche Fehler zu beheben.

Der mögliche bevorstehende Wechsel zu ARM trägt ebenfalls dazu bei, da keine dieser älteren Technologien jemals auf ARM existiert hat. Nun hätte Apple mit dem Fallenlassen von 32-Bit noch so lange warten können, bis der Wechsel zu ARM sie endgültig dazu gezwungen hätte; das hätte es wohl für manche Menschen einfacher gemacht, die Gründe zu verstehen. Allerdings hätte es bedeutet, die Entwicklung von macOS längere Zeit aufzuhalten.

Rückwärtskompatibilität ist letztendlich ein Kostenfaktor: Da beinhaltet Kosten für die Wartung älterer Technologien, Kosten, die neuere Technologien zu verlangsamen, Kosten durch eine größere Oberfläche für Angriffe. Es gibt keine richtige Antwort darauf, auf wie hohe Kosten man sich als Entwickler einlassen sollte. Microsoft hat sich dafür entschieden, die Kosten in Kauf zu nehmen und eine umfassende Rückwärtskompatibilität zu bieten, da deren Vorteile ihm wertvoller sind als die Kosten. Apple verfolgt da eher einen Mittelweg, indem es die Rückwärtskompatibilität bis zu einem gewissen Punkt beibehält, aber letztlich es immer wieder bevorzugt, regelmäßig veraltete Technologien zu bereinigen, um voranzukommen.

Hoffentlich hat Ihnen dies einen tieferen Einblick in die Art und Weise gegeben, wie Apple im Laufe der Jahre mit Technologie-Übergängen und Abwärtskompatibilität umgegangen ist. Es mag zwar den Anschein haben, dass Apple sich wenig um Abwärtskompatibilität kümmert, aber sie verbringen viel Zeit damit, die Vor- und Nachteile abzuwägen, bevor sie die Unterstützung für irgendetwas fallen lassen. Es ist auch interessant, wie viel Mühe sie in die Abwärtskompatibilität investiert haben, wenn man bedenkt, was für große tiefgreifende Veränderungen sie in den letzten 30 Jahren unternommen haben, auch wenn die Abwärtskompatibilität nur eine vorübergehende ist.