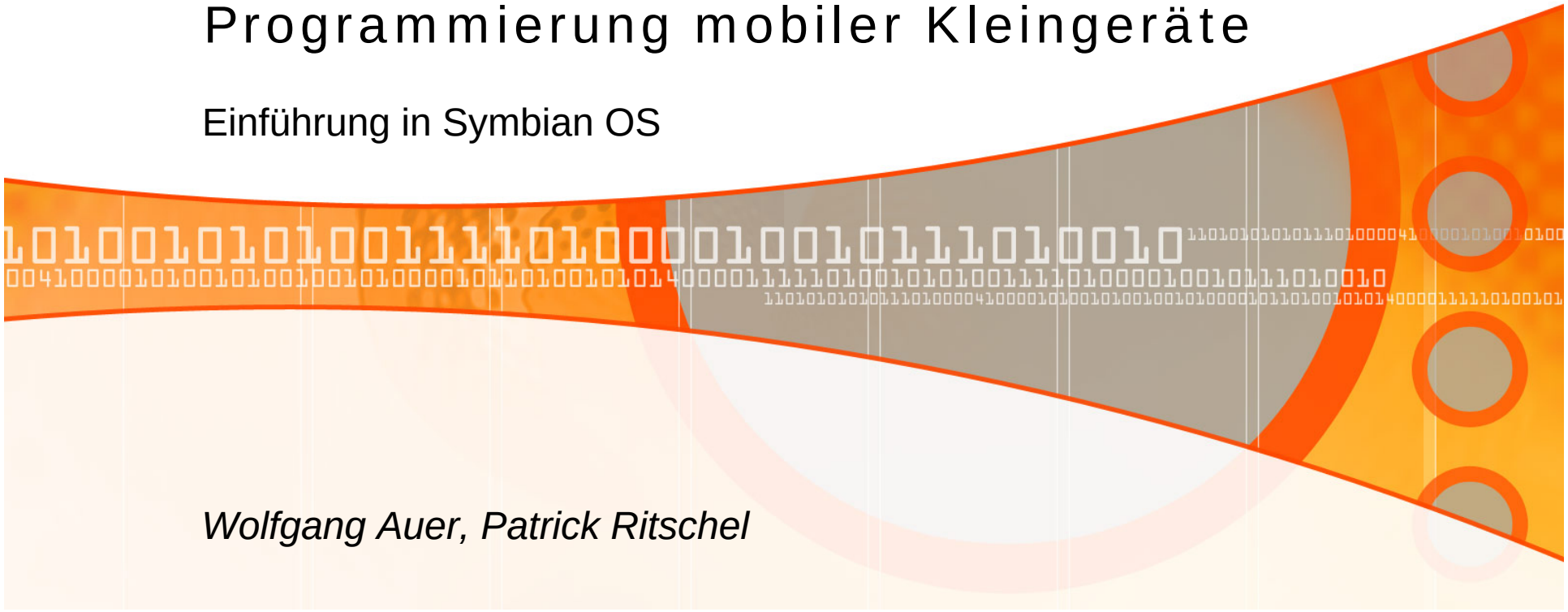
Programmierung mobiler Kleingeräte

Einführung in Symbian OS

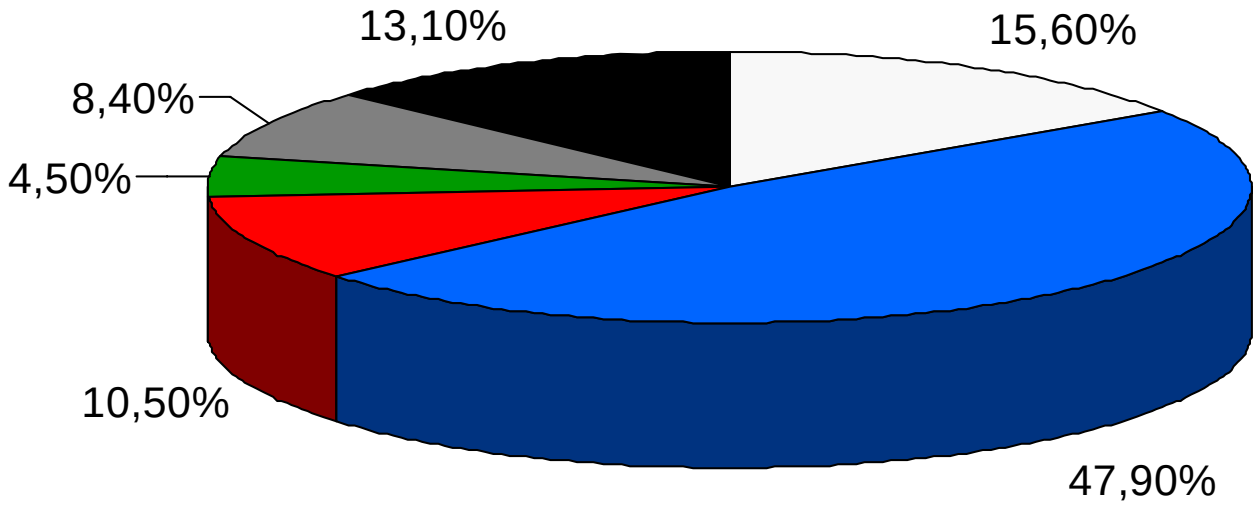


Wolfgang Auer, Patrick Ritschel

Was bzw. Wer ist *Symbian*?

> *"Symbian is the market leading provider of open OS software with an 85 percent share of smartphone market to date"*
> *Quelle: Symbian Fast Facts bezugnehmend auf Gartner*

- Firmenpartner



Entwicklung von Symbian OS
Plattformen

- *Symbian OS* basiert auf Psion *EPOC* Release 5
- Mit *Symbian OS* entstanden unterschiedliche Plattformen, die sich nur hinsichtlich GUI unterscheiden

Plattform

Version	***Series 60***	***Series 80***	***Series 90***	***UIQ***	***FOMA***
V 6.0/6.1	**1st Edition** Nokia 7650	Nokia 9210			Foma D702i
V 7.0(s)	**2nd Edition** Nokia 6600		Nokia N90	Sony Ericsson P800/P900	
V 8.0/8.1	Nokia N70				
V 9.1/9.2	**3rd Edition** Nokia N80			Sony Ericsson M600i	

Binärkompatibilität zu älteren Versionen nicht mehr gegeben

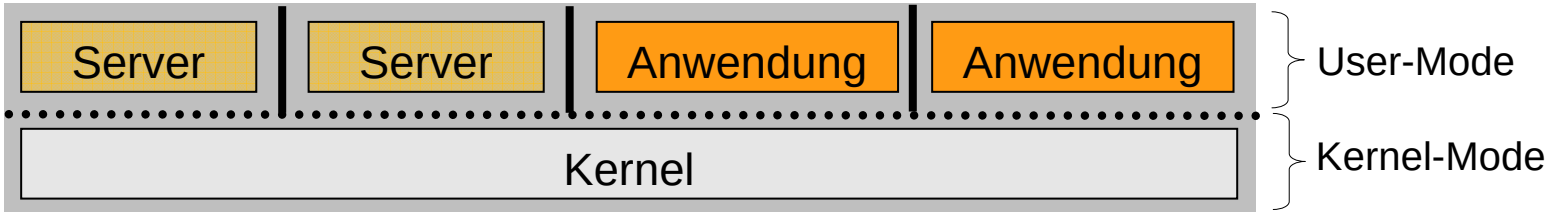
Designziele von *Symbian OS*

- Betriebssystem für mobile Kleingeräte
  - Große Vielfalt von Geräten
  - Ressourcen-Sparende (Ereignis-Gesteuerte, effizientes Power-Management, ...)
  - Massenmarkt (Robustheit, Lokalisation, ...)
- Hohes Maß an Robustheit
  - Drahtlose Kommunikation wechselnder Qualität
  - Ressourcenbeschränkung
  - Massenmarkt (Systemausfall)
- Offene Plattform für Gerätehersteller **und** Softwareentwickler

Hardware

- CPU
  - 32-Bit mit niedrigen Taktfrequenzen (36-, 52-, 104- oder 156Mhz-ARM bzw StrongARM-CPUs)
- Speicher
  - ROM
    - min 18 MB (V9.1, S60 komprimiert) für Betriebssystem, Basismiddleware und Standardanwendungen
    - Standardprogramme werden direkt im ROM ausgeführt
  - RAM
    - min 10 MB (V9.1, S60) als Arbeitsspeicher für laufende Programme und Speicher für Daten und Anwendungen
- Ein- und Ausgabegeräte

Architektur (1)
Komponenten



Prozessgrenze

Privilegiengrenze

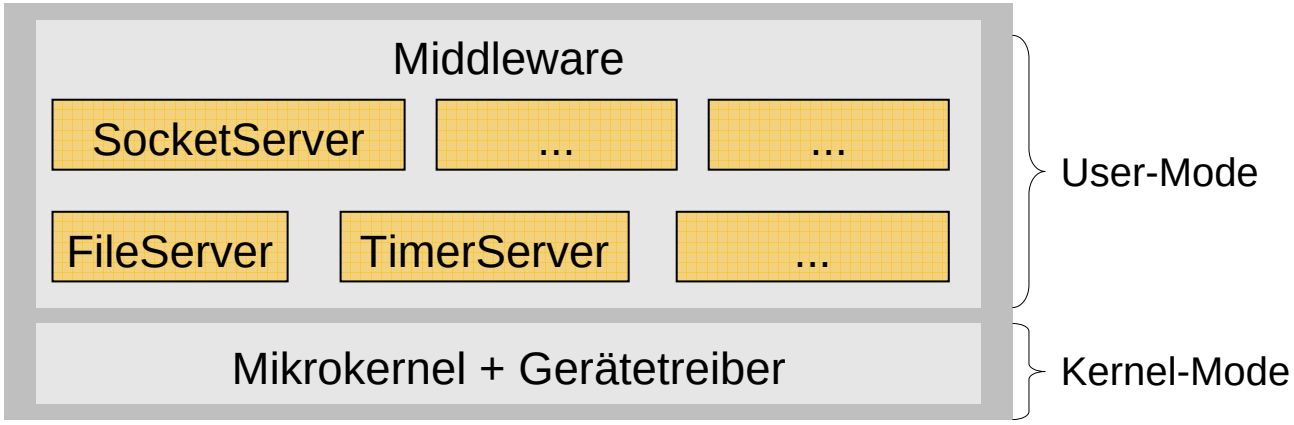
- *Kernel*
  - verwaltet z.B. CPU, Speicher, E/A-Geräte, ..
  - kann über privilegierte Anweisungen direkt auf HW-Komponenten zugreifen
  - regelt Zugriff andere SW-Komponenten auf diese Hardwarekomponenten
- Server
  - User-Mode-Anwendung ohne Benutzerschnittstelle
- Anwendung
  - User-Mode-Anwendung mit Benutzerschnittstelle

Architektur (2)
Mikrokernel

- Speichersparender Kernel (~300kB), der nur die notwendigsten Funktionen zur Verfügung stellt
  - Speichermanagement
  - Gerätetreiber
  - Powermangement
- Ab OS 8.0 wird ein Real-time Kernel (EKA2) verwendet
- Vorteile:
  - Robustheit
  - kurze Aufstartzeit
  - kurze Antwortzeiten auf "Low-Level"-Events (Tastatur, ..)

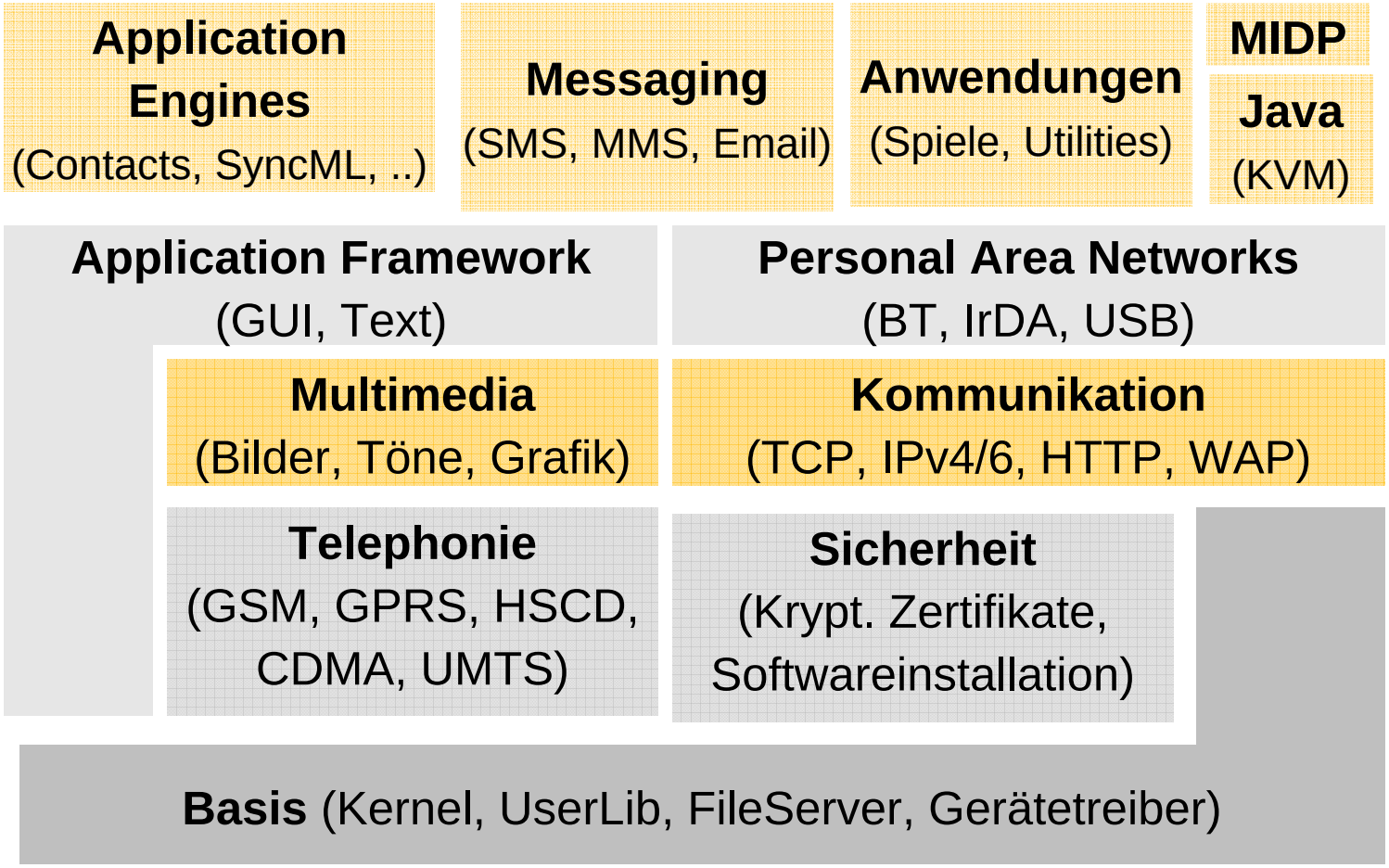
Architektur (3)
Server-Komponenten

- Server-Komponenten, die sich außerhalb des Kernels befinden, stellen die zusätzliche Middleware dar.
  - Kommunikation
  - Dateisystem, ...
- Server-Komponenten laufen in User-Mode

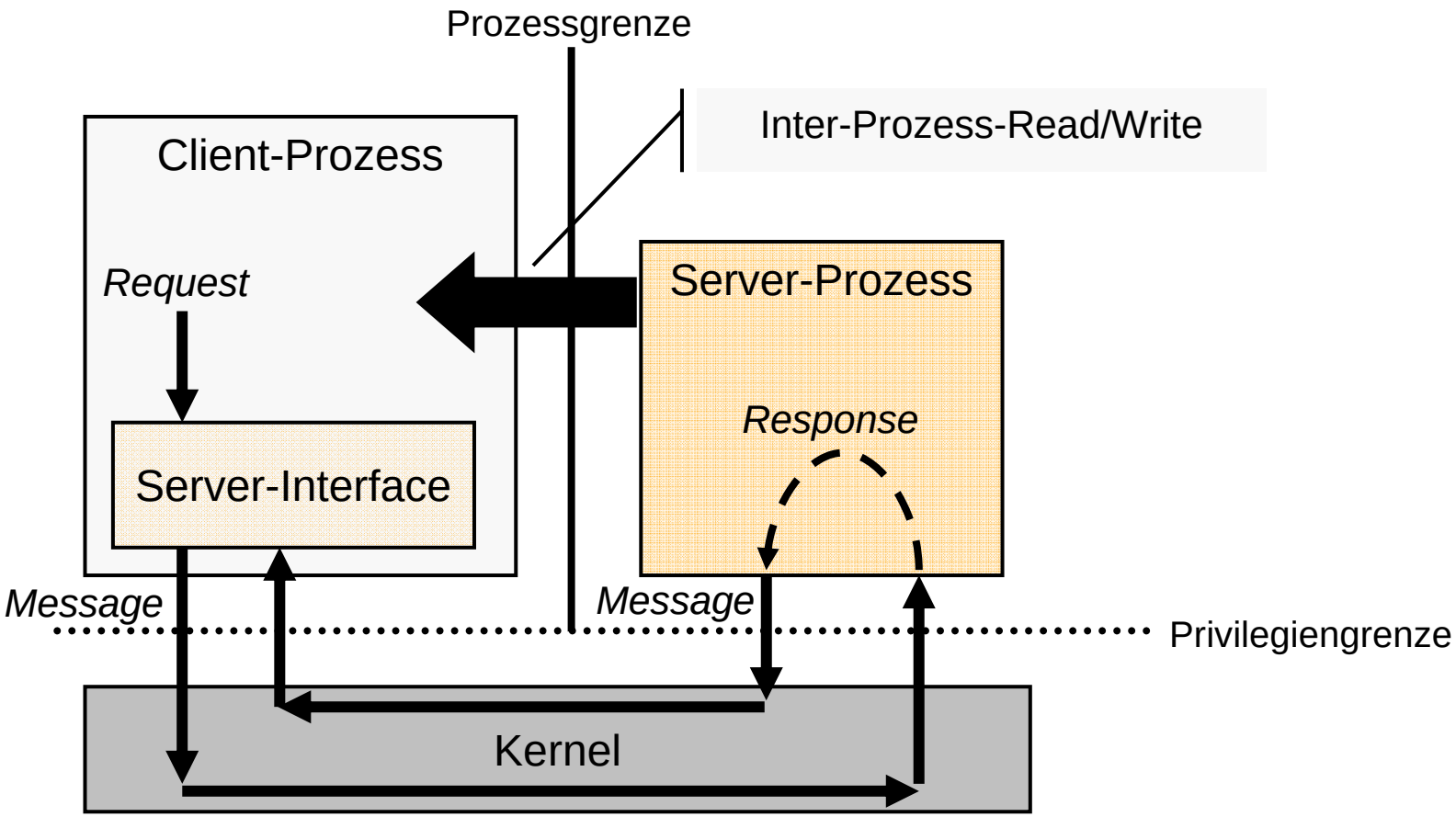


Architektur (4)
Systemkomponenten

Struktur von Symbian OS



Client/Server Architektur



Speicherverwaltung

- ROM
  - besteht aus Dateien, die im Laufwerk Z verwaltet werden
  - Informationen sind direkt auf Adressen abgebildet
    - einfacher Zugriff
    - direkte Ausführung von Programmen
- RAM
  - 4k-Blöcke
  - Virtueller Adressraum von *User*-Prozessen und Kernel-Server-Prozessen
  - RAM-Disk, die auf das Laufwerk C angebildet wird
  - DLLs, die nicht aus dem ROM geladen werden
  - Mapping Table der MMU

Speicherverwaltung
Adressraum eines Prozess

- Systemübergreifender Speicher: ROM und shared DLLs im RAM
- Prozessübergreifender Speicher: Code, statische beschreibbare Daten
- Stack (default **12kB**) und Heap (in der Regel 2MB)

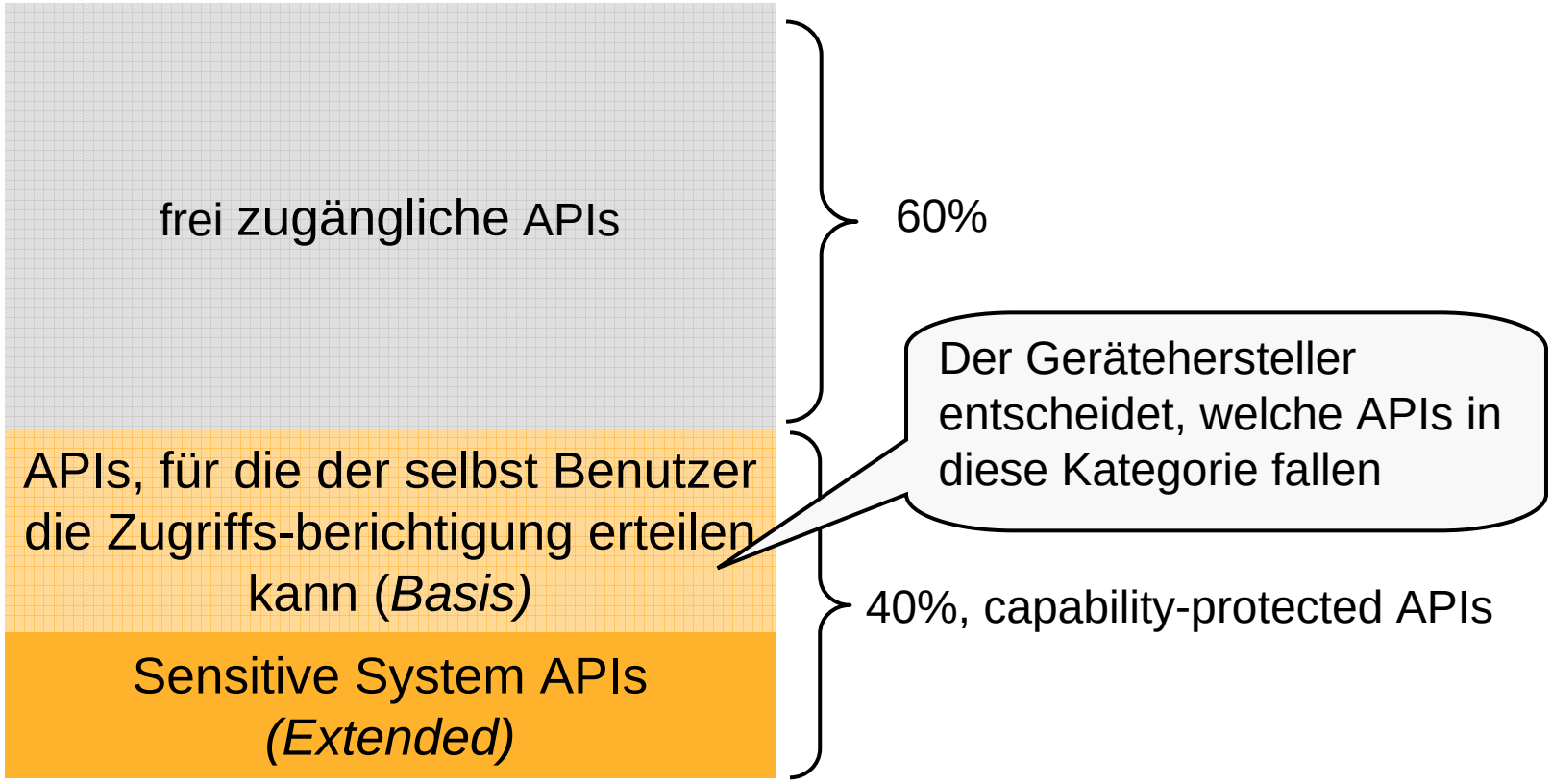
Sicherheit
Platform Security

- OS 8.x
  - Sicherheitsprüfung bei der Installation der Applikation
  - Nach Installation voller Zugriff auf Dateisystem und alle APIs
  - Zertifizierung von Applikationen (Symbian Signed)
- OS 9
  - mit OS 9.x wurde ein erweitertes Sicherheitsmodell "PlatSec" eingeführt (Zur Laufzeit)
    - Schutz vor unerlaubtem Zugriff auf Hardware, Software und Daten
    - "Data caging": Abschottung der privaten Daten einer Applikation. (SID)
    - Schutz vor fehlerhafter bzw. schlecht implementierter Software

Sicherheit
Vertrauen und Befähigung (*Capability*)

- *Berechtigungen werden immer für einen Prozess erteilt d.h. Vertrauen wird einem Prozess ausgesprochen*
- *Befähigungen (*capabilities*) beziehen sich auf das Vertrauen, das einem Prozess ausgesprochen wird*
  - *Befähigungen werden für ausführbare Code zum Build-Zeitpunkt abhängig von den benötigten APIs festgelegt*
  - *Zur Laufzeit werden diese Befähigungen angefordert*
  - *Befähigungen eines Prozesses können sich nicht ändern*
  - *Vergabe und Prüfung von Befähigungen ist für Endanwender weitestgehend transparent*

Sicherheit
Einteilung der Symbian OS APIs



Sicherheit
Capability-protected APIs

- Capability-protected APIs
  - 40% aller Symbian OS APIs, der Rest ist nach wie vor frei zugänglich!
  - nur **autorisierter** Code darf auf diese APIs zugreifen
- Einteilung der der capability-depended APIs
  - Basis
    - Zugriff auf Benutzerdaten, Informationen über Benutzerumgebung, Zugriff auf Netzwerk
    - Rechte werden durch explizite Nachfrage beim Benutzer durch signierten Code gewährt
  - Erweitert
    - Zugriff auf low-level Funktionalität z.B. Änderung der Netzwerk-konfiguration, Power-Management, ...
    - Erweiterte Tests werden vorgenommen
    - Zugriff nur über signierten Code
  - Spezielle
    - Trusted Computer Base TCB: Uneingeschränkter Zugriff auf Hardware und Software

Sicherheit
Autorisierung

- Symbian Signed
  - Sicherheitssiegel für Symbian Software, die nach bestimmten Kriterien entwickelt, von einem speziellen Testzentrum getestet und von Symbian zertifiziert wurde
- Unsigned-Sandboxed
  - alle ungeschützten APIs + Basis capability-dependent APIs
- Berechtigung
  - Blanket grant: Der Benutzer erteilt die Berechtigung bei der Installation der Applikation. Diese gilt bis zur Deinstalltion der Applikation
    - z.B. Erlaubnis für den Zugriff auf die Benutzerdaten
  - Single shot grant: Der Benutzer muss jeden einzelnen Zugriff erlauben
    - z.B. Netzwerkzugriffe, falls die Applikation nicht "Symbian Signed" ist

Namenskonventionen (1)

- Namenskonventionen spielen eine wichtige Rolle. Anhand der Konventionen kann erkannt werden,
  - in welchem Speicherbereich ein Objekte einer bestimmten Klasse angelegt werden soll.
  - ob in einer Funktion etwas schief gehen kann und man dementsprechend Vorkehrungen treffen muss.
  - ob es sich um einen asynchronen Dienst handelt.
  - ...

Namenskonventionen (2)

- Klassenname

Präfix	Bedeutung
C	Klasse, die auf dem Heap angelegt werden soll und von `CBase` abgeleitet ist. `CBuf,...`
T	Einfacher Wertetyp, der keine externen Referenzen zu anderen Objekten besitzen. `TInt, TBool, ...`
R	Klassen, die Referenzen auf Ressourcen, die irgendwo anders gehalten werden, besitzen. RSocket, RFile, ...
M	Interface Klassen, die ein abstraktes Protokoll definieren, das die Unterklassen implementieren müssen.

- Datenkomponenten von Klassen: `i<Name>`
- Parameter von Methoden: `a<Name>`
- Konstante: `K<Name>`

Namenskonventionen (3)

- Postfix bei Methodennamen

Postfix	Bedeutung
L	In der Methode kann ein *Leave* auftreten ⇒ Verwendung von CleanupStack notwendig, falls zuvor dynamisch Speicher angelegt wurde
LC	Methode liefert als Resultat ein dynamische angelegtes Objekt zurück, das in der Methode bereits auf den CleanupStack gelegt wurde. `CS* pS = CS::NewLC(p);`

LD	Methoden, die die Verantwortung für das jeweilige Objekt übernehmen und für die notwendige Freigabe des Speichers sorgt
	`CEikDialog* pDlg = new (ELeave) CSettingsDialog();`
	`pDlg->ExecuteLD(R_SETTINGS_DIALOG)`

Basistypen

- Symbian verfügt für alle skalaren Datentypen Varianten `TInt, TFloat, TChar`, ...
- Um Plattformunabhängigkeit zu garantieren, soll die Verwendung von C/C++-Typen vermieden werden.
- `TChar`: 32 bit-Integerwert, Unicode
- `TBool`: 32 bit-Integerwert
- `TAny` statt `void`

Fehlerbehandlung und Ressourcen-Management (1)

- Strategien von Symbian
  - Fehlertolerante Konstruktoren und Konventionen
  - Fehlerbehandlung für Funktionen
    - Funktion wird auf jeden Fall ordnungsgemäß verlassen und anschließend der Fehler behandelt (vergleiche Ausnahmebehandlung)

- *Trap-Leave-Mechanismus, CleanupStack* und *Konventionen*, der zur Vermeidung von Speicherlöchern und offenen Ressourcen verwendet wird.

Fehlerbehandlung und Ressourcen-Management (2)

- *Trap-Leave*-Mechanismus
  - Zur Entstehungszeit von Symbian war der *C++ -Exception-Mechanismus* noch inadäquat
  - *Trap-Leave-Mechanismus* ist für Symbian OS optimiert und versucht weniger Aufwand als der Standard Exception-Mechanismus
- TRAP bzw. TRAPD: Vergleichbar try/catch-Blöcken:
  `TRAPD(errCode, doExampleL());`
- `User::Leave(KErrCode)`: wie throw-Anweisung

Fehlerbehandlung und Ressourcen-Management (3)

- *CleanupStack* dient dazu,
  - sich "Kopien" von lokalen Zeigern auf dynamischen Speicher zu halten
  - sich Referenzen für beanspruchte Ressourcen, zu halten
- Im Fall eines Fehlers, werden alle Zeiger und Referenzen, die **ab dem letzten TRAP-Handler** auf den CleanupStack gelegt worden sind, abgearbeitet d.h. Speicher deallokiert und Ressourcen zurückgegeben.

Fehlerbehandlung und Ressourcen-Management (4)

- Verwendung des *CleanupStacks*

```
CConsoleBase* pConsole = Console::NewL(KConsoleTitle,
     TSize(KConsFullScreen,KConsFullScreen));
CleanupStack::PushL(pConsole);
...
RSocketServ socketServer;
User::LeaveIfError(socketServer.Connect());
CleanupClosePushL(socketServer);

//initialize the socket
RSocket socket;
User::LeaveIfError(socket.Open(socketServer, KAfInet,
                               KSockStream, KProtocolInetTcp));
CleanupClosePushL(socket);
...
CleanupStack::PopAndDestroy(&socket); //close socket (resource)
CleanupStack::PopAndDestroy(&socketServer);
CleanupStack::PopAndDestroy(pConsole); //free allocated memory
```

Sicherstellen, dass auch bei Leave Consolen- Objekt gelöscht wird!

Auch bei einem Leave muss die Ressource wieder freigegeben werden

Programmierung in C++

Fehlerbehandlung und Ressourcen-Management (5)

- Hinweise zur Verwendung des *CleanupStacks*
  - Datenkomponenten eines Objekts, die dynamische angelegt werden, dürfen nicht auf dem CleanupStack abgelegt werden, da dies auf jeden Fall im Destruktor zerstört werden.
  - Zeiger und Referenzen müssen in umgekehrter Reihenfolge, in der Sie auf den CleanupStack gelegt würde wieder heruntergeholt werden Der Parameter bei `PopAndDestroy(&socket)` dient nur dem leichteren Debuggen.

Fehlerbehandlung und Ressourcen-Management (6)

- Objekterzeugung
  - new-Operator wurde überladen `new (ELeave)`
    - Falls keine Speicher vorhanden ist, wird ein Leave ausgelöst.
    - Datenkomponenten werden automatisch mit Standardwert initialisiert
  - Für komplexe Objekte wird eine *Zwei-Phasen-Erzeugung* (Factory-Pattern) angewendet

```
class CFoo : public CBase {
   private:
       CFoo();
       constructL();

   public:
       static CFoo* NewL();
       static CFoo* NewLC();
};
```

  - Konstruktoren dürfen kein Leave erzeugen!

Fehlerbehandlung und Ressourcen-Management (7)

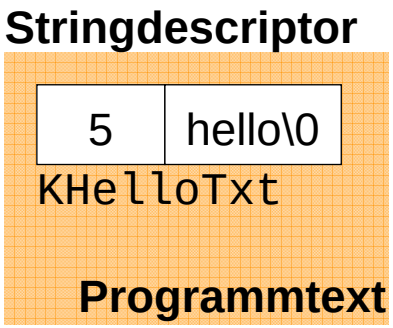
- Objektzerstörung
  - Destruktoren dürfen nicht von vollständig initialisieren Objekten ausgehen
  - Nach der Zerstörung einer Datenkomponente muss diese immer auf NULL gesetzt werden

```
delete ipName;
ipName = NULL;
ipName = aName.AllocL();
```

Wir auf das Null-Setzen vergessen und AllocL löst ein Leave aus, kann über ipName immer noch auf den bereits freigegebenen Speicher zugegriffen werden

Descriptoren (1)

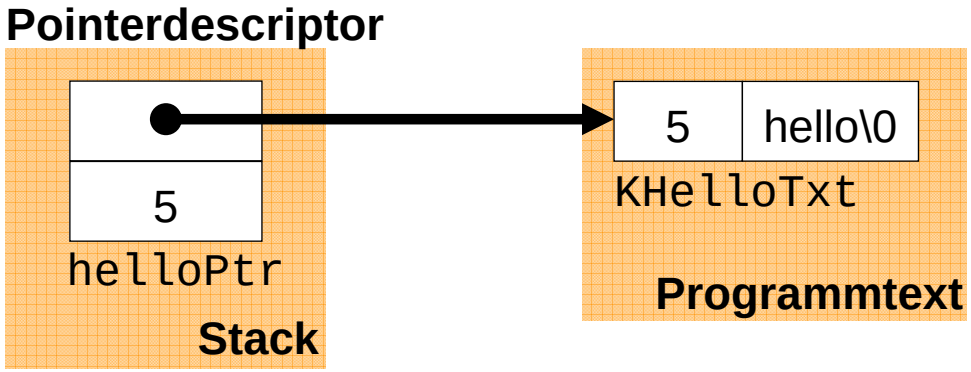
- *Descriptoren* erlauben den konsistenten Umgang mit Text und Binärdaten
  - Komfortabler, sicherer als C-Strings
  - Volle Kontrolle über Speicher
- Descriptoren können Ihre Daten am Stack, Heap oder auch im Programmtext ablegen




```
_LIT(KHelloTxt, "hello");
```

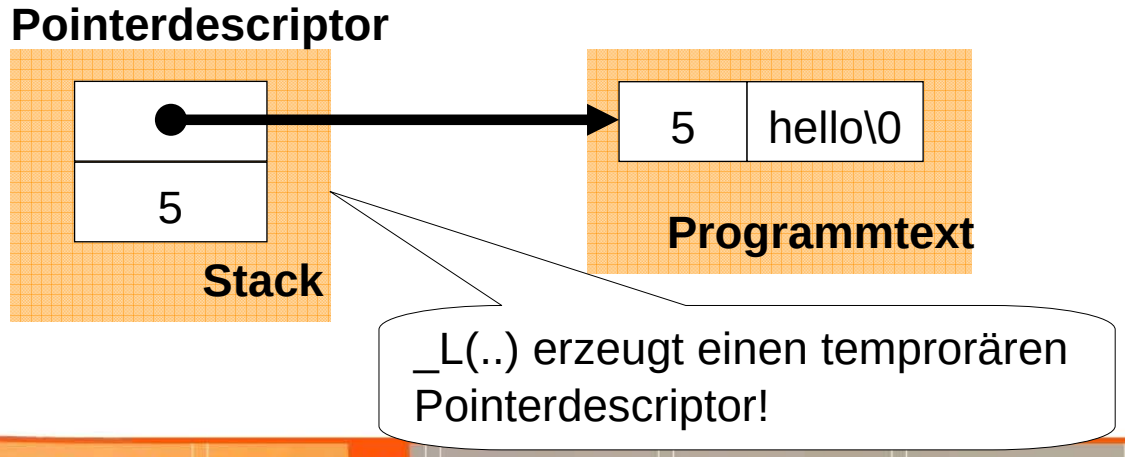
Es wird die Variable KHelloTxt vom Typ **TLitC<TInt>** definiert und mit dem Zeichenkettenliteral *hallo* initialisiert.

Descriptoren (2)

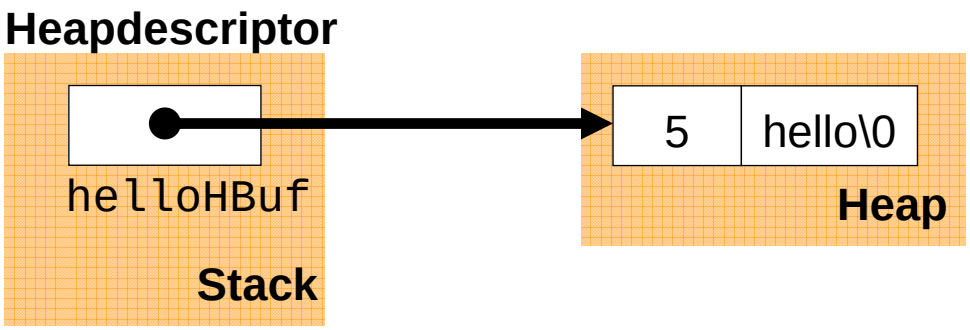



```
TPtrC helloPtr = KHelloTxt;
```

```
_L("Hello"); //deprecated
```

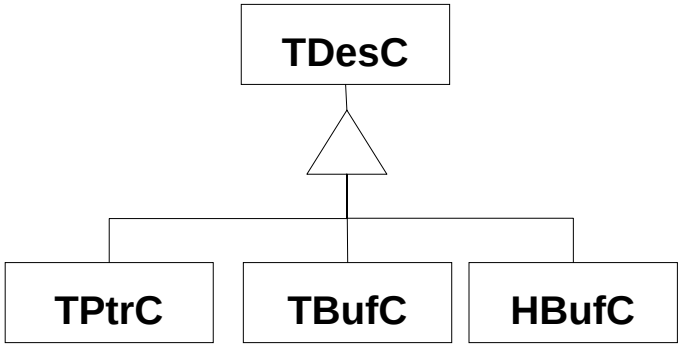


Descriptoren (3)



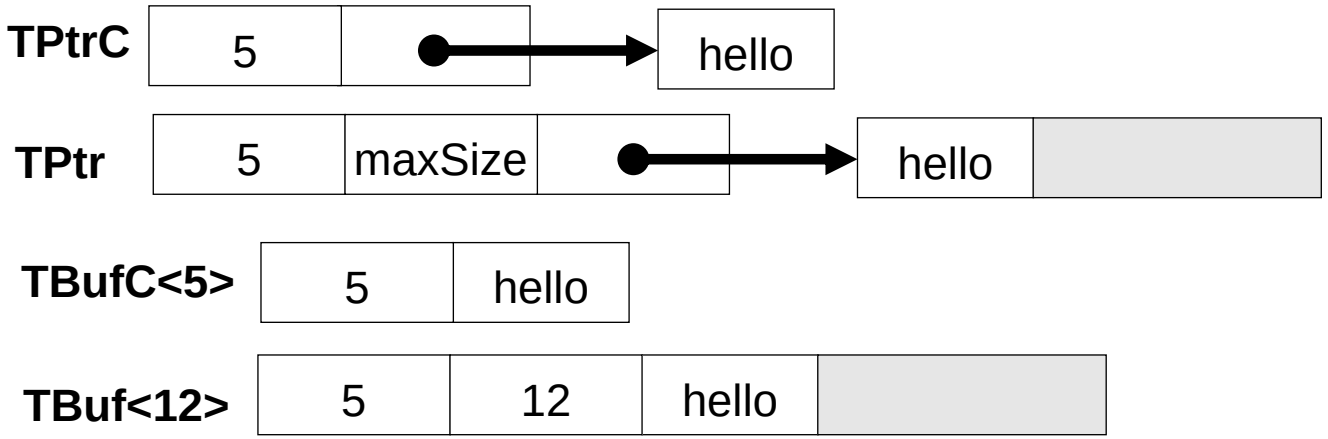

```
HBufC* helloHBuf = KHelloTxt().AllocLC();
```

- Klassenhierarchie für konstante Descriptoren



Descriptoren (4)

- Unterschied konstante und veränderbare Descriptoren



- Für HeapDescriptoren gibt es nur `HBufC`. Änderungen an den Inhalten können über TPtr vorgenommen werden

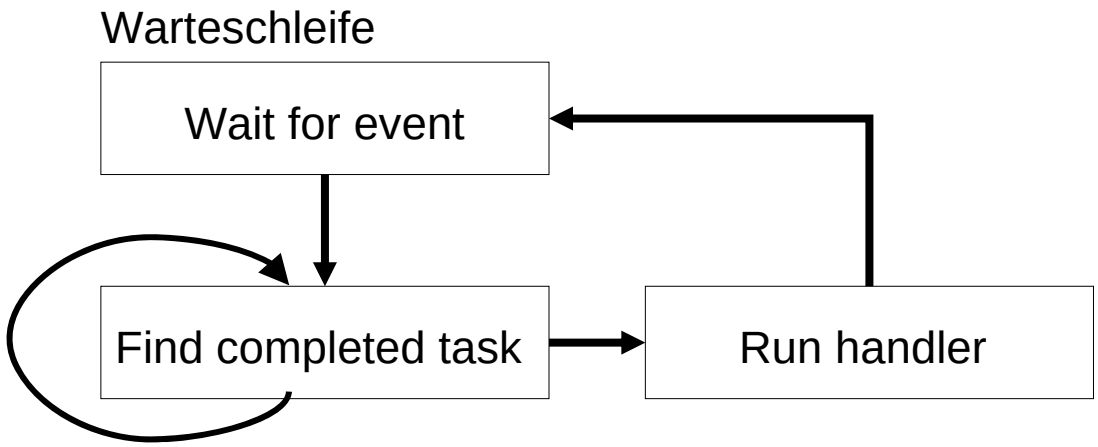
```
HBufC* helloHBuf = KHelloTxt().AllocLC();
TPtr ptr = buf->Des();
```

Für alle Decriptoren gibt es eine 8 bzw. 16bit-Variante z.b.: `TPtr8` bzw. `TPtr16`.

Asynchronität (1)

- Nebenläufigkeit und Asynchronität spielen in modernen Betriebssystemen eine wichtige Rolle ⇒ Multitasking und Event-Handling
- Symbian OS bietet preemptives Multitasking und Multithreading.
- Aus Effizienzgründen wird aber von der Verwendung von Multithreading zur Realisierung von Asynchronität abgeraten ⇒ kooperatives Multitasking

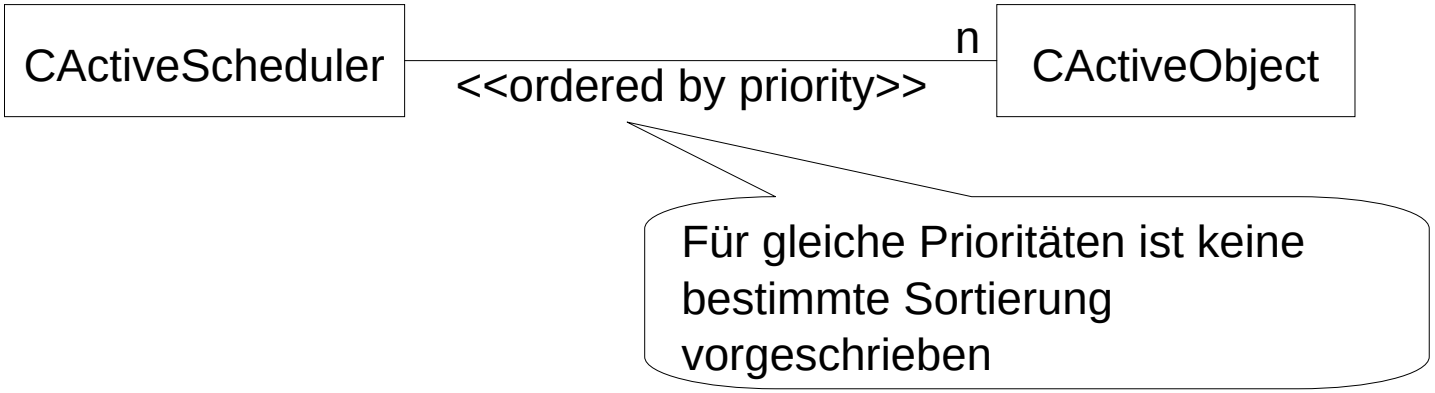
Asynchronität (2)
Kooperatives Multitasking



- Die Warteschleife startet und blockiert den Applikationsthread
- Sobald ein Request abgearbeitet worden ist, wird ein Signal an die Warteschleife gesendet und es wird eine neue Aufgabe gesucht und gegebenenfalls gestartet
- Die Kommunikation zwischen Applikationsthread und Service Provider der den Request bearbeitet, erfolgt über eine Semaphore
- Signale werden in einer Queue verwaltet.

Asynchronität (3)
CActiveScheduler

- `CActiveScheduler` implementiert die Warteschleife und die zusätzlich benötigten Funktionen.



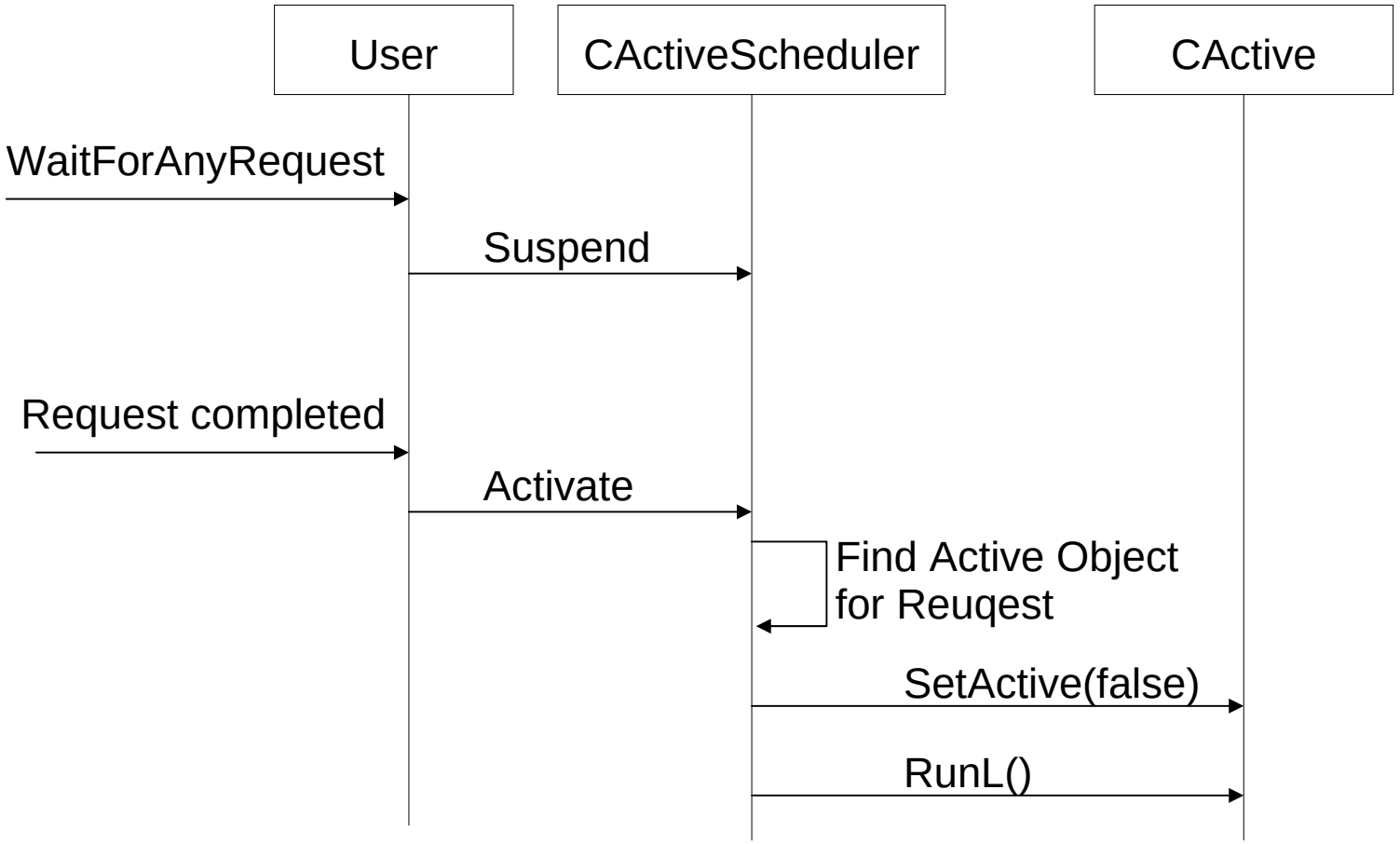
- `User::WaitForAnyRequest()` suspendiert den Applikationsthread bis ein asynchroner Request abgearbeitet worden ist

Asynchronität (4)
CActiveScheduler

- Entweder das Application-Framework bietet bereits eine Instanz der Klasse `CActiveScheduler` an oder es muss explizit eine Instanz angelegt und installiert werden.

```
CActiveScheduler* pActiveScheduler
        = new (ELeave) CActiveScheduler();
CleanupStack::PushL(pActiveScheduler);

CActiveScheduler::Install(pActiveScheduler);

//make sure there is at least one outstanding request
CElementsEngine* pElemEngine = CElementsEngine::NewLC(*pConsole);
pElemEngine->LoadFromCsvFilesL();

CActiveScheduler::Start();
CleanupStack::PopAndDestroy(2, pActiveScheduler);
```

Asynchronität (5)
CActiveScheduler



Programmierung in C++

Asynchronität (6)
CActive

```
class CActiveObjectExperimentsAppUi;

class CTimeProfligator : public CActive {
   private:
      RTimer iTimer;

   private:
      CTimeProfligator(TInt aPriority);
      void ConstructL();

      virtual void RunL();
      virtual void DoCancel();

   public:
      static CTimeProfligator* NewLC(TInt aPriority = EPriorityStandard);
      static CTimeProfligator* NewL(TInt aPriority = EPriorityStandard);

      virtual ~CTimeProfligator();

      void Start(…);
};
```

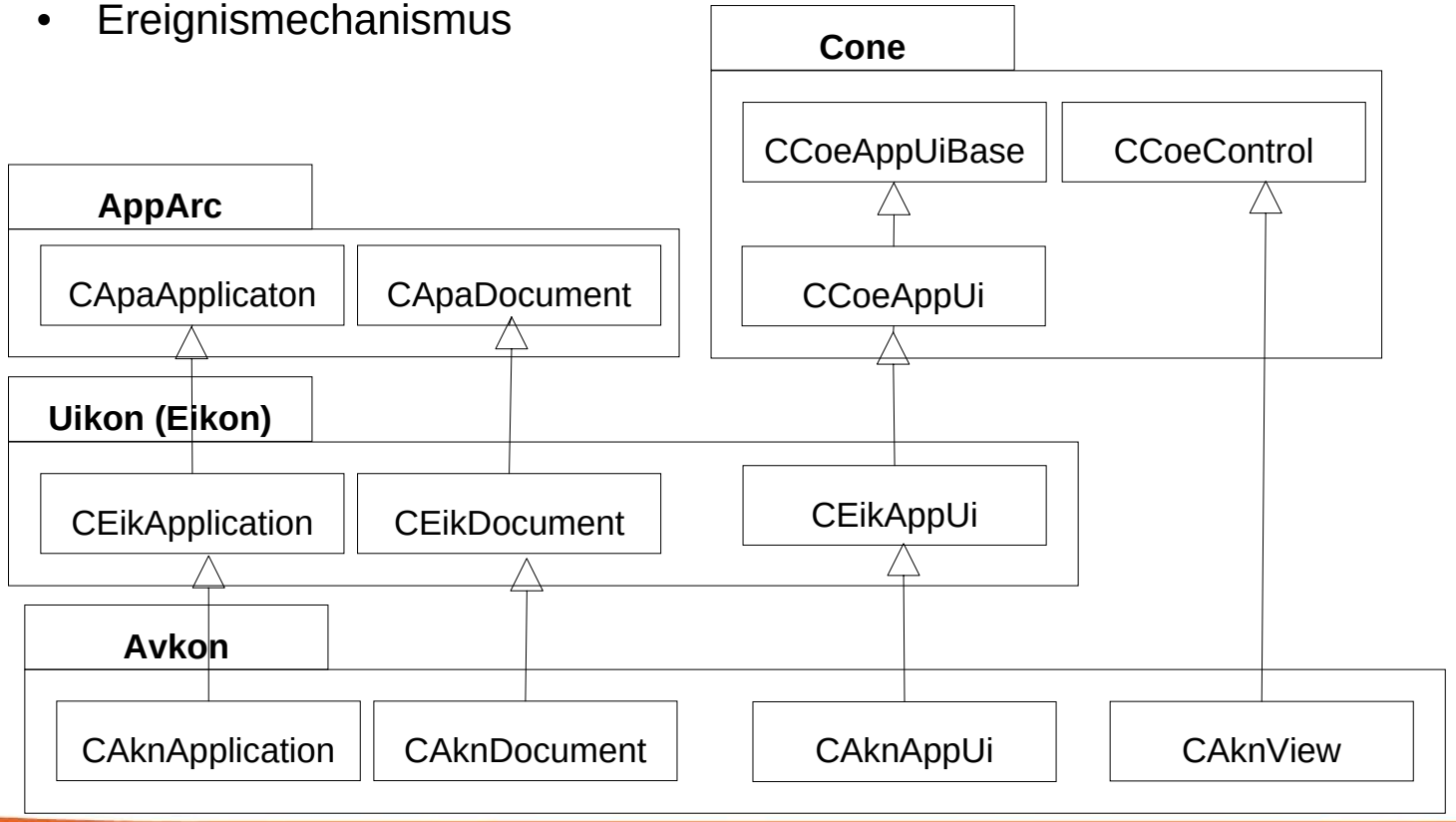
Programmierung in C++

Asynchronität (7)
CActive

```
CTimeProfligator::~CTimeProfligator() {
   Cancel();
   iTimer.Close();
}

void CTimeProfligator::ConstructL() {
   User::LeaveIfError(iTimer.CreateLocal());
   CActiveScheduler::Add(this);
}

void CTimeProfligator::RunL() {/* request completed*/}
void CTimeProfligator::DoCancel() { iTimer.Cancel(); }

void CTimeProfligator::Start(…) {
   if (IsActive()) { //starting a new timing event
                     //while still one is outstanding causes panic
      Cancel();
   }

   iTimer.AfterTicks(iStatus, aProfligationTicks);
      SetActive();
}
```

Applikationsdesign (1)
Applikation Framework

- Series 60 bietet ein vorgefertigtes Applikationsframework, in dem bereits die gesamte Basisfunktionalität von GUI-Applikationen abgebildet ist.
  - Fenster, Steuerelemente
  - Ereignismechanismus

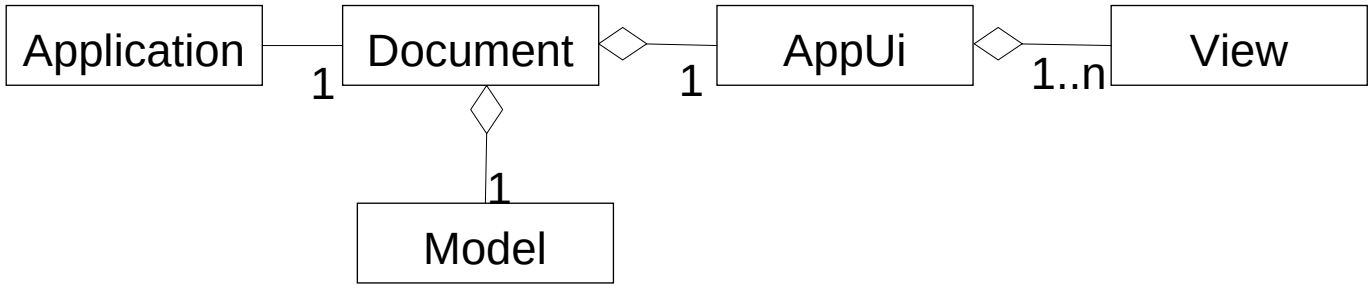


Applikationsdesign (2)
Applikation Framework

- *AppArc* (Application Architecture)
  - Ausgangspunkt für die Document/View
- *Cone* (Control Environment)
  - Mechanismen für Benutzereingaben und die Darstellung der Benutzerschnittstelle
- *Uikon* (Eikon)
  - Generische, geräte-unabhängige Implementierung von allgemeinen Klassen. Zusätzlich bietet Uikon eine UI-Bibliothek, die auf allen Symbian System identisch ist.
- *Avkon*
  - Series 60 spezifische Implementierung der GUI-Bibliothek.

Applikationsarchitektur (1)
Document/View

- Es wird eine Document/View-Architektur verwendet



- *Application:* Einstiegspunkt z.B. E32Dll()..
- *Document:* Verwaltung der Applikationsdaten
- *AppUI:* Controller, der Ereignisse selber abhandelt oder an die entsprechende View weiterleitet
- *View:* Repräsentation der Modelldaten, etc.

Applikationsarchitektur (2)

- Designmöglichkeiten, die vom Framework unterstützt werden:
  - Control-Based Architecture
  - Dialog-Based Architecture
  - Avkon View-Switching Architecture

- Control-Based und Dialog-Based sind im wesentlichen identisch. Die Unterschiede liegen in den Controls, die als Basis für die View verwendet wird.
  - Control-Based: Views, die als Container bezeichnet werden, werden direkt von `CCoeControl` angeleitet
  - Dialog-Based: Views sind Unterklassen von `CAknDialog`.

- Avkon View-Switching Architecture verwendet einen zentralen View-Server, bei dem Unterklassen von `CAknView` registriert werden können

Applikationsarchitektur (3)
Welchen Ansatz für welche Zwecke?

- Control-Based:
  - Controls können beliebig auf einer Pane positioniert werden.
  - Scrolling, schließen der View, etc. muss explizit implementiert werden
  - Kein Layoutmanagement!
  - Gute geeignet, wenn nur eine View benötigt wird
  - Wenn Controls enthalten sind mit hohen Ansprüchen an ihre Privatsphäre

- Dialog-Based:
  - Applikation besteht aus einer Aneinanderreihung von Dialogen.
  - Layout wird automatisch erstellt. Informationen über Controls werden aus Ressource-Dateien gelesen
  - Gut geeignet für Applikationen, die nur zur Eingabe von Daten oder Einstellungen dienen.

Applikationsarchitektur (4)
Welchen Ansatz für welche Zwecke?

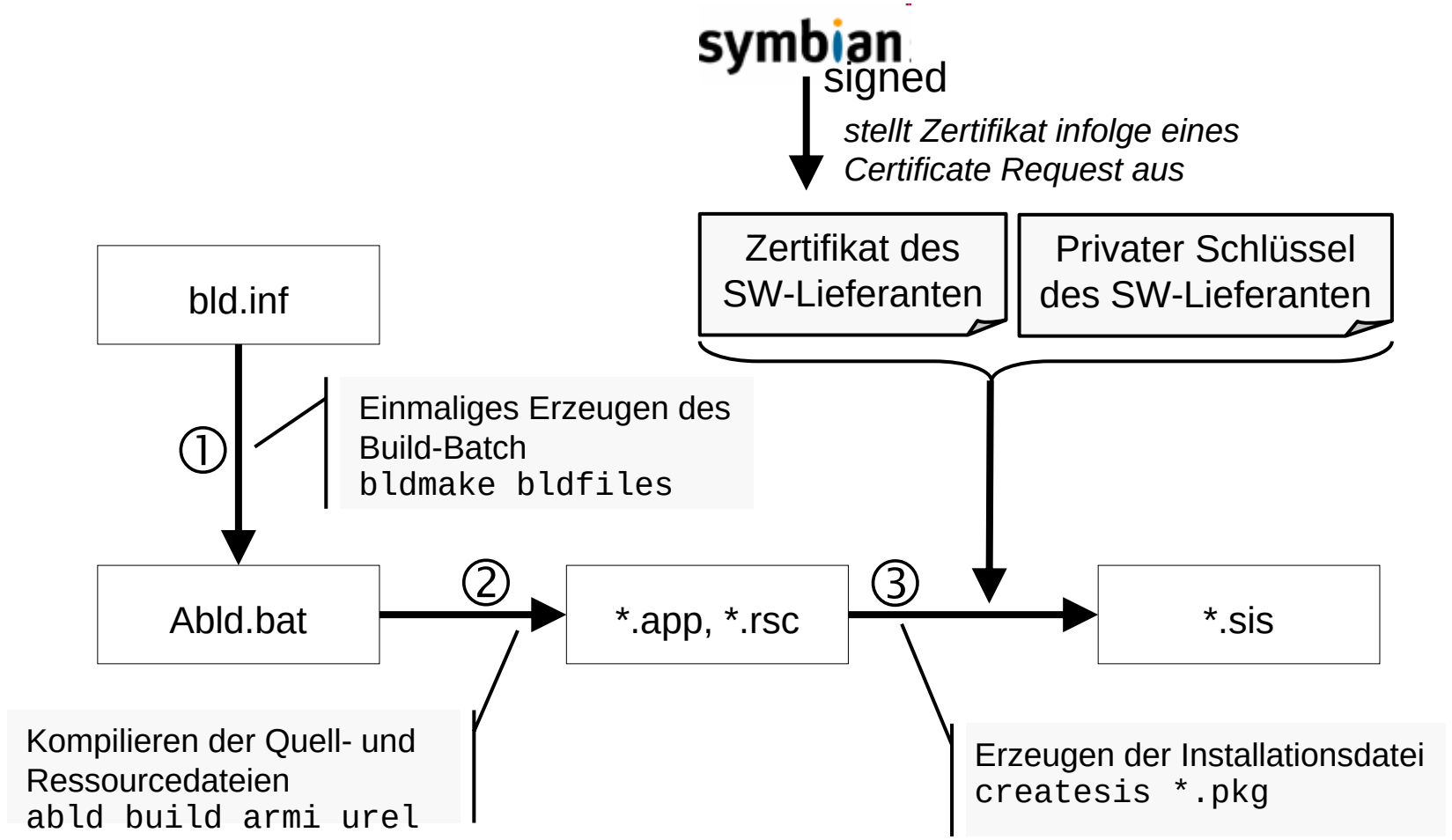
- Avkon View-Switching
  - Applikation wird nicht mehr als eigenständiger Teil gesehen, sondern integriert sich lückenlos in das Gesamtsystem. Der Wechsel zwischen den Applikationen fällt im Idealfall gar nicht mehr auf.

Entwicklungsprozess (1)
Beteiligte Dateien

- Quellcode
- Projektdefinitionsdatei (*.mmp*)
  - Applikationstyp (app, dll)
  - UID2: Spezifikation der Zielplattform (Version)
  - UID3:
    - eindeutige Identifizierung der Applikation (Versionisierung)
    - **Sicherheit** (Beachte festgelegte Bereiche hinsichtlich Symbian Signed!) (OS 9.x)
  - Verweis auf Quelldateien, Ressourcendateien, benötigte Bibliotheken
  - CAPABILITY: Benötigte Befähigungen (OS 9.x)
- Ressourcendateien, die Informationen für die grafische Benutzeroberfläche enthalten
- Package-Datei (*.pkg*), in der Inhalt eines Installationspaketes für das Endgerät definiert wird
- Zertifikat und privater Schlüssel des Entwicklers (OS 9.x)

Entwicklungsprozess (2)

Build-Prozess für ARM



Referenzen, Quellen, etc.

Literatur

- Symbian OS, Eine Einführung in die Anwendungsentwicklung, A. Gerlicher, Stephan Rupp, dpunkt.verlag
- Developing Series 60 Applications*, L. Edwards*, R. Barker, Addison Wesley

Onlinereferenzen

- `www.symbian.com`
- `www.newlc.com`
- `www.forum.nokia.com`

Referenzen