Bluetooth

Markus Merath, Marcus Koehler, Stephan Helten, Marc Seeger

Sommersemester 2007

Praktikum Mobile Applications, HdM Stuttgart

Einführung

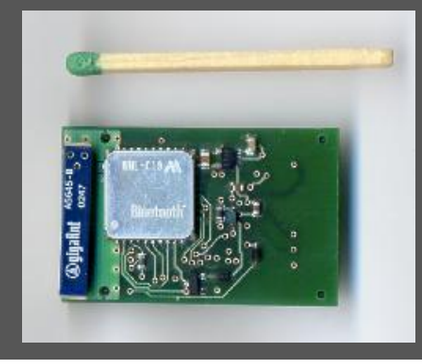
Bluetooth

„ … eine offene Spezifikation für drahtlose Übertragung von Daten und Sprache“

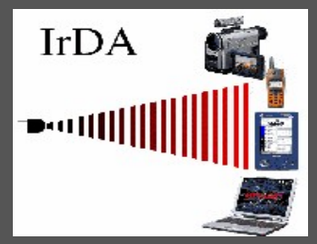
Fujitsu Siemens Computers

Was ist Bluetooth?

- Möglichkeit zur drahtlosen Datenübertragung
- Es handelt sich um ein Personal Area Network (PAN)
- Übertragung von Daten und Sprache
- Drahtlose und gesicherte Verbindung zwischen den Geräten
- Kein proprietäres Protokoll
- „Kabelersatz“
- Preisgünstige Lösung „Ein-Chip-Lösung“



Vergleich zu Infrarot



- Das Einsatzgebiet ist mit der Infrarot-Lösung (IrDA) vergleichbar
- Man benötigt keine Sichtverbindung
- Bluetooth hat eine größere Reichweite als IrDA
- Es besteht die Möglichkeit, dass mehr als zwei Geräte miteinander in Verbindung stehen können, im Gegensatz zu IrDA, bei der sich die Kommunikation auf eine Point-to-Point Verbindung beschränkt.

Anwendungsmöglichkeiten

- Hauptsächlich Mobiltelefone

- Lautsprecher und Kopfhörer

- Spielkonsolen (Nintendo WII)

- Drucker, Mäuse & Tastaturen

- Digitale Kameras

Geschichte

- Die Firma Ericsson initiierte 1994 eine Studie.
- Gründung der SIG (**S**pecial **I**nterest **G**roup) 1998
- Namensgeber: Harald Gormsen
- Offizielle Ankündigung von Bluetooth am 20. Mai 1998

Entwicklung

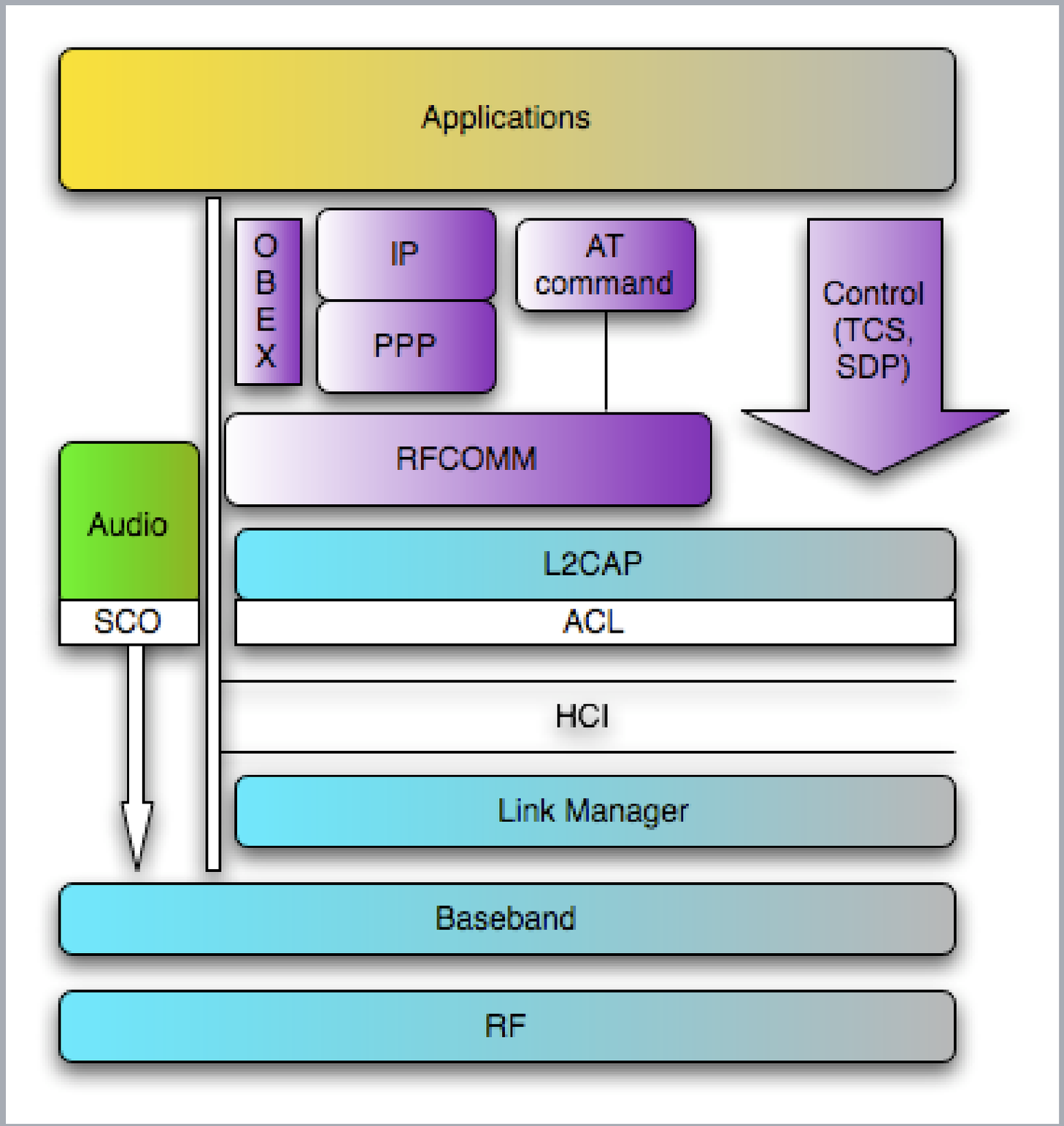
- Bluetooth Version 1.0a (**Juni 1999**), Version 1.0b (**Dezember 1999**)
- Bluetooth-Geräte erschienen im **Jahr 2000**
- Standard 802.15 wurde von der IEEE verabschiedet
- Bluetooth Version 1.1 (**Dezember 2000**)
- Bluetooth Version 1.2 (**November 2003**)
- Bluetooth Version 2.0 + EDR (**November 2004**)

Bluetooth Protokollstapel

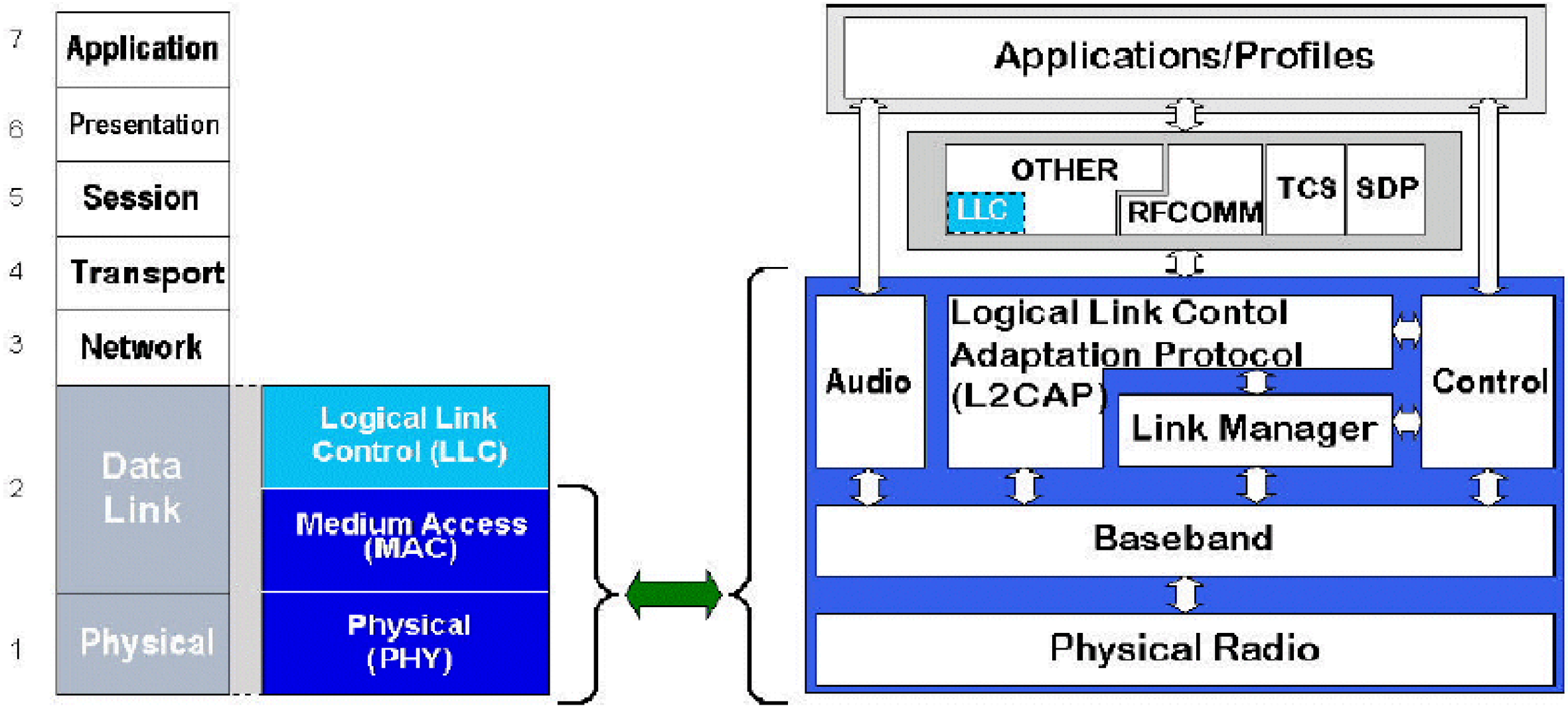
Der Bluetooth Protokollstapel

1. Core Spezifikationen
2. Funkschicht
3. Profile Spezifikationen
4. Basisband
5. Verbindungsverwaltung und -steuerung (LMP,L2CAP)
6. Dienstfindung (SDP)
7. Middlewareschicht (RFCOMM,HCF, Profile)

Der Protokollstapel



Einordnung WPAN in OSI



1. Die Core Spezifikation

Kern Spezifikation umfasst mehrere Teile:

Teil A	legt Funkspezifikation fest
Teil B	enthält die Basisbandspezifikation und spezifiziert den Link-Controller, welcher z.B. die Basisbandprotokolle ausführt
Teil C	enthält das Linkmanager-Protokoll für den Verbindungsaufbau (LMP)
Teil D	ist die Logical Link Control and Adaption Protocol Spezifikation. (L2CAP)
Teil E	definiert das Service Discovery Protokoll (SDP)
Teil F	befasst sich mit Schnittstellen zu seriellen Bausteinen, IdDA, Telefon,WAP-Diensten
Teil H	befasst sich mit technischen Schnittstellen wie USB
Teil I	beschreibt Testumgebungen

2. Funkschicht

Diese Schicht behandelt:

- die Funkübertragung
- genutzte Frequenzen
- Modulationen und Sendeleistungen

- Die Funkschicht entspricht im wesentlichen der Bitübertragungsschicht des OSI- bzw. 802-Modells.
- Die Schicht behandelt die Funkübertragung und Modulation der Signale.

3. Profile Spezifikationen

- Enthält Protokolle und Funktionen zur Anpassung von Bluetooth an herkömmliche uns neue Anwendungen

4. Basisband

Die Basisbandschicht beschreibt die Mechanismen:

- zum Verbindungsaufbau
- die Rahmenstruktur
- und das Zeitverhalten.

- Die Schicht entspricht in etwa der MAC-Teilschicht des 802-Standards (Sicherungsschicht des OSI-Modells)
- Enthält aber auch Elemente der Bitübertragungsschicht.

5. Verbindungsverwaltung und -steuerung

- Link Manager Protocol, LMP
- Logical Link Control and Adaption Protocol, L2CAP
- Diese Schicht besteht aus Protokollen, die lose miteinander in Verbindung stehen.
- Der Link Manager steuert den Verbindungsaufbau und die Verbindungsverwaltung zwischen zwei Geräten, inklusive Sicherheits- und Authentifizierungsfunktionen.
- Das L2CAP ist ein Protokoll der Sicherungsschicht, das die höheren Schichten an die Fähigkeiten des Basisbandes anpasst und Übertragungsdetails (verbindungslos und verbindungsorientiert) verbirgt.

6. Dienstfindung

- Hierfür wird das Service Discovery Protokoll (SDP) verwendet
- Das SDP dient zur Erkennung und Suche nach Diensten mit bestimmten Eigenschaften und der Beschreibung von Diensten innerhalb der Funkreichweite eines Bluetooth-Gerätes.

7. Middlewareschicht

Weitere Schichten bestehen aus einer Mischung verschiedener Protokolle welche oft auch unter der Bezeichnung „Middleware-Schicht“ zusammengefasst werden.

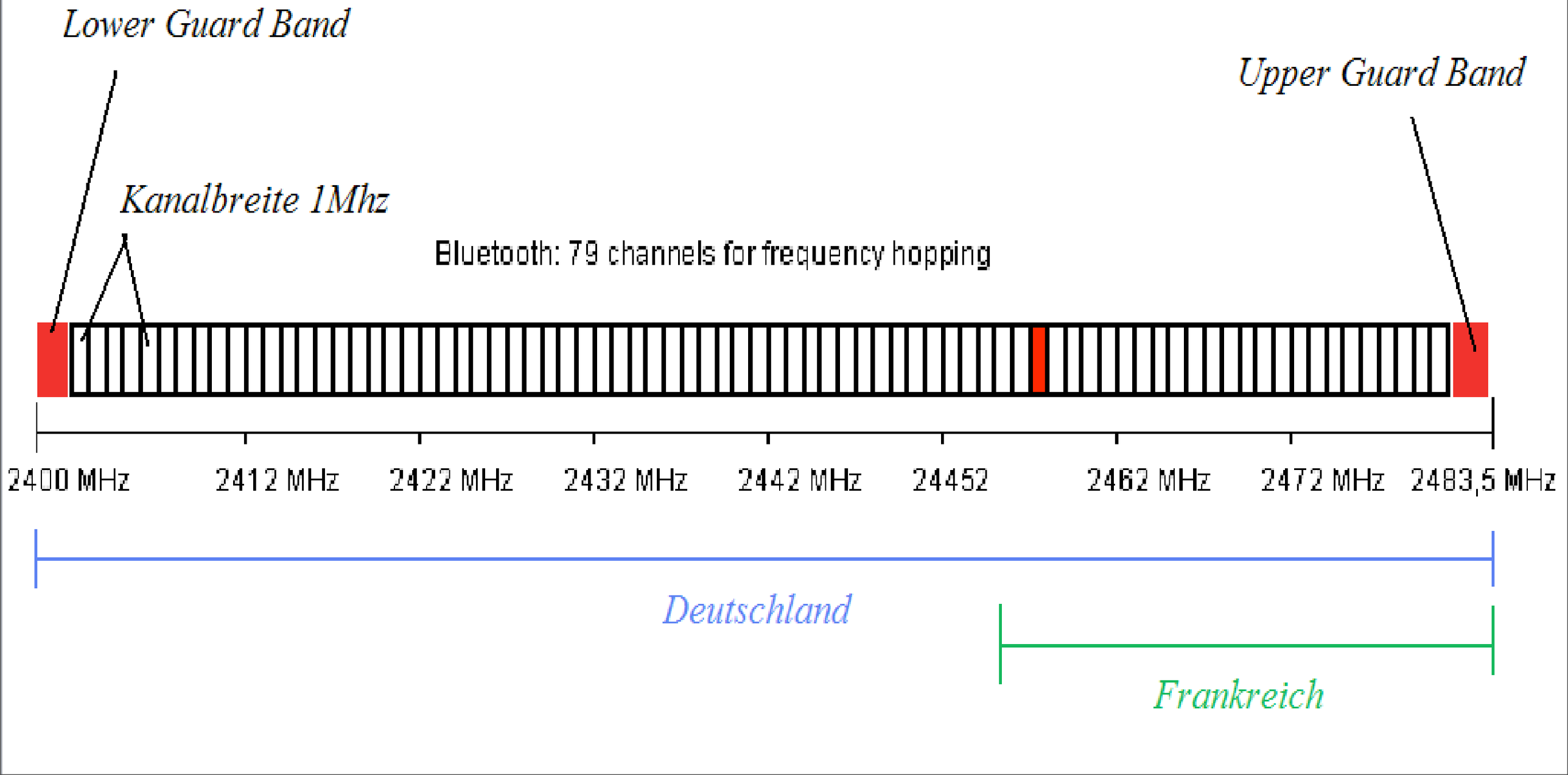
- Das RFCOMM Protokoll Emulation einer seriellen Schnittstelle
- Das TCS BIN Protokoll dient zur Sprachübertragung
- Die HCI-Schnittstelle bietet Zugang zum Basisband
- TCP/IP / PPP / BNEP dient den Internetanwendungen
- (OBEX) zum Austausch von Kalenderinformationen und Visitenkarten

Bluetooth Funkschicht

Einteilung

(1) Trägerfrequenz
(2) Frequency Hopping Spread Spectrum (FHSS)
(3) Das Piconetz
(4) Das Scatternetz
(5) Modulationsverfahren
(6) Leistungsklassen

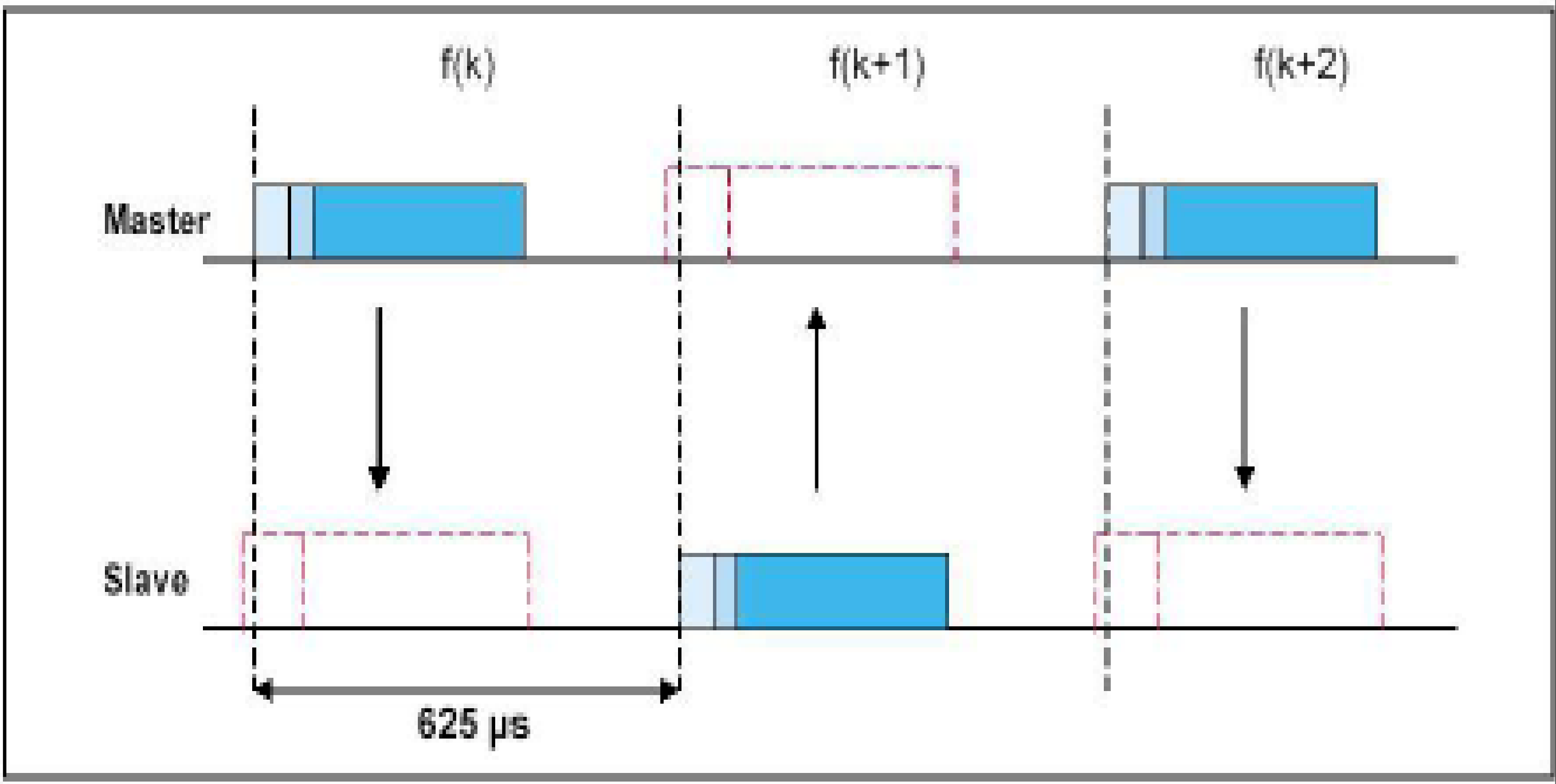
Die Trägerfrequenz



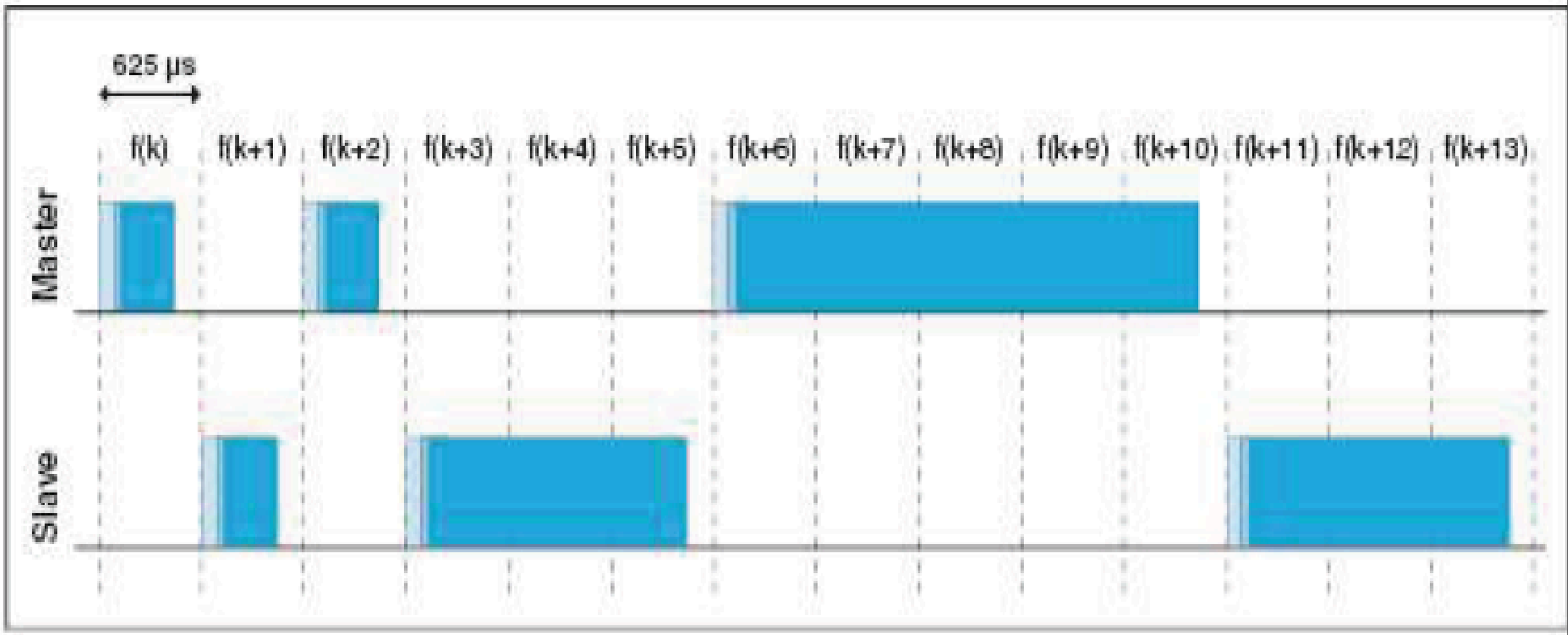
Frequency Hopping Spread Specturm (FHSS)

- Übertragung erfolgt mit einem kombinierten Frequenzsprung/Zeitduplexverfahren, dem FHSS
- Aufteilung des Kanals in sogenannte Slots, die 625µs lang sind
- Jeder Slot nutzt eine andere (Hop)frequenz
- Ein Generator für Pseudozufallszahlen erzeugt die Folge von Frequenzen
- Jede Frequenz ist nur für eine gewisse Zeitspanne aktiv (mindestens 625µs)
- Zuordnung der Slots durch das TDD-Verfahren

Beispiel 1

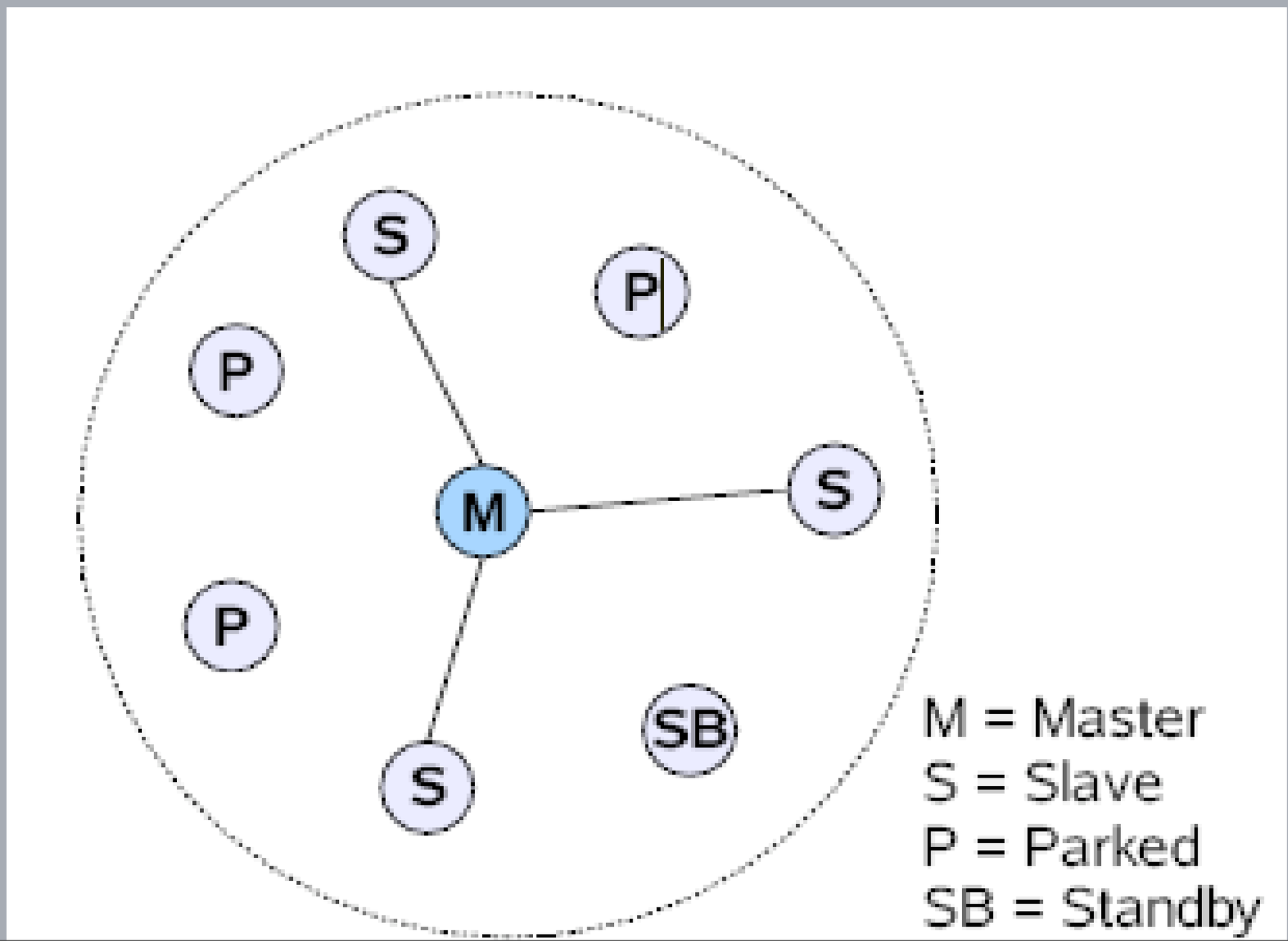


Beispiel 2


625 µs
f(k)
f(k+1)
f(k+2)
f(k+3)
f(k+4)
f(k+5)
f(k+6)
f(k+7)
f(k+8)
f(k+9)
f(k+10)
f(k+11)
f(k+12)
f(k+13)
Master
Slave

Das Pikonetz

- Die Grundeinheit eines Bluetooth Systems



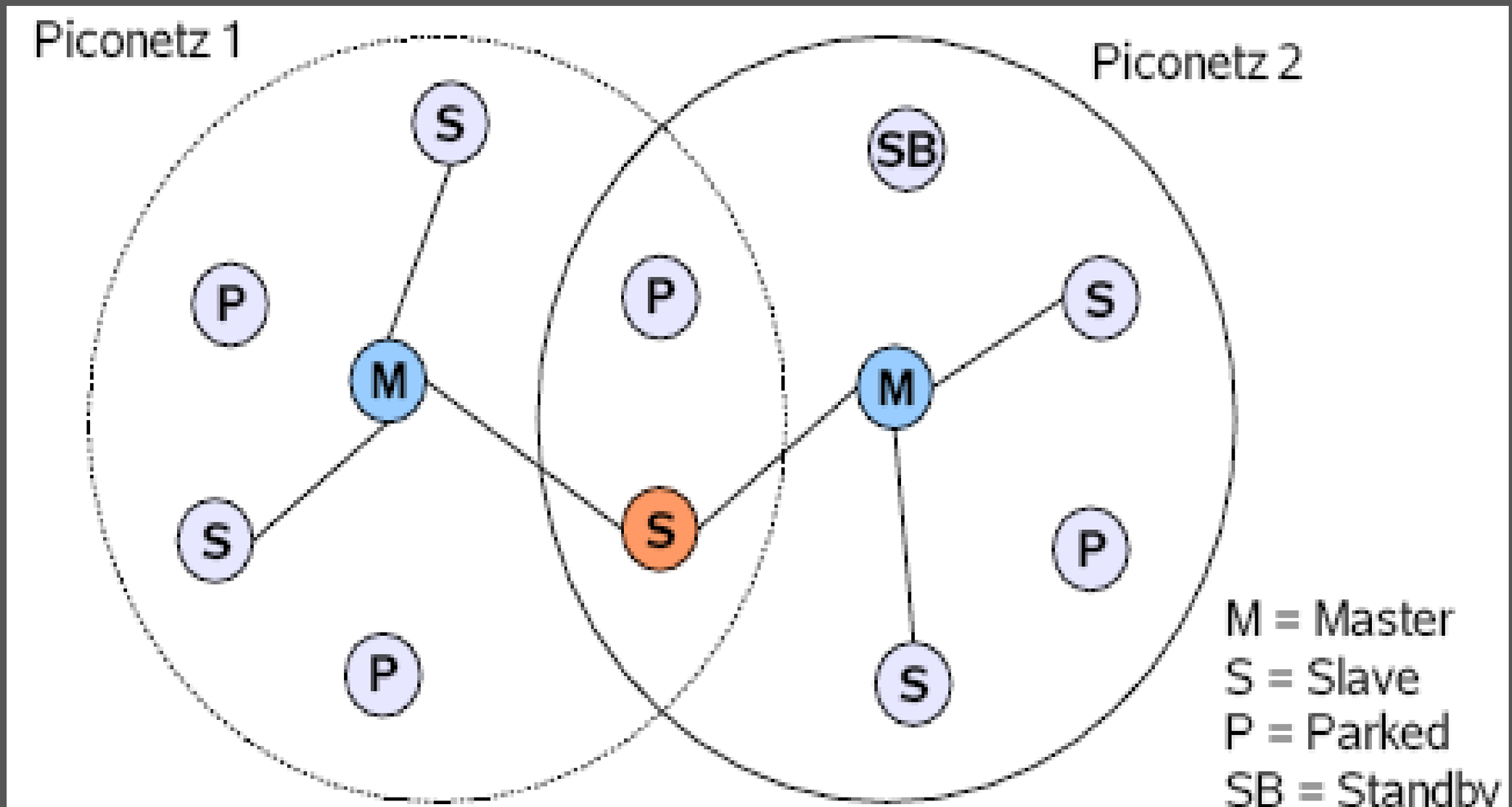
Bildung eines Pikonetzes

- Schritte zur Bildung eines Pikonetzes:
  - Synchronisation – Master muss seine Gerätekennung und den Wert seiner inneren Uhr aussenden
  - Gerät, dass Pikonetz einrichtet wird zum Master
  - Zur Teilnahme muss der Slave lediglich seine Uhr an die des Masters anpassen.

→ Problem?!

Das Scatternetz (Streunetz)

Bild zeigt ein Scatternet mit 2 Piconetzen



Modulationsverfahren

Übertragungsgeschwindigkeit	Brutto	Netto	Modulation
Basic Rate	1 MBit/s	723,2 kBit/s	GFSK
Enhanced Data Rate	2 MBit/s	1446,2 kBit/s	π/4DQPSK
	3 MBit/s	2169,6 kBit/s	8DPSK

- Bluetooth-Version 1.0 – 2.0 benutzen die Modulationsart GFSK in der Basic-Rate
- Bluetooth Version 2.0 noch zusätzlich π/4DQPSK und 8DPSK, dadurch können höhere Übertragungsraten erzielt werden.

Leistungsklassen

Einteilung der Bluetooth-Geräte in verschiede Leistungsklassen (Power Classes)

Leistungsklasse	Max. Leistung	Nominale Leistung	Minimale Leistung	Reichweite[9]
1	100 mW	N/A	1mW	100 – 150 m
2	2,5 mW	1mW	0,25mW	10 – 25 m
3	1 mW	N/A	N/A	10 m

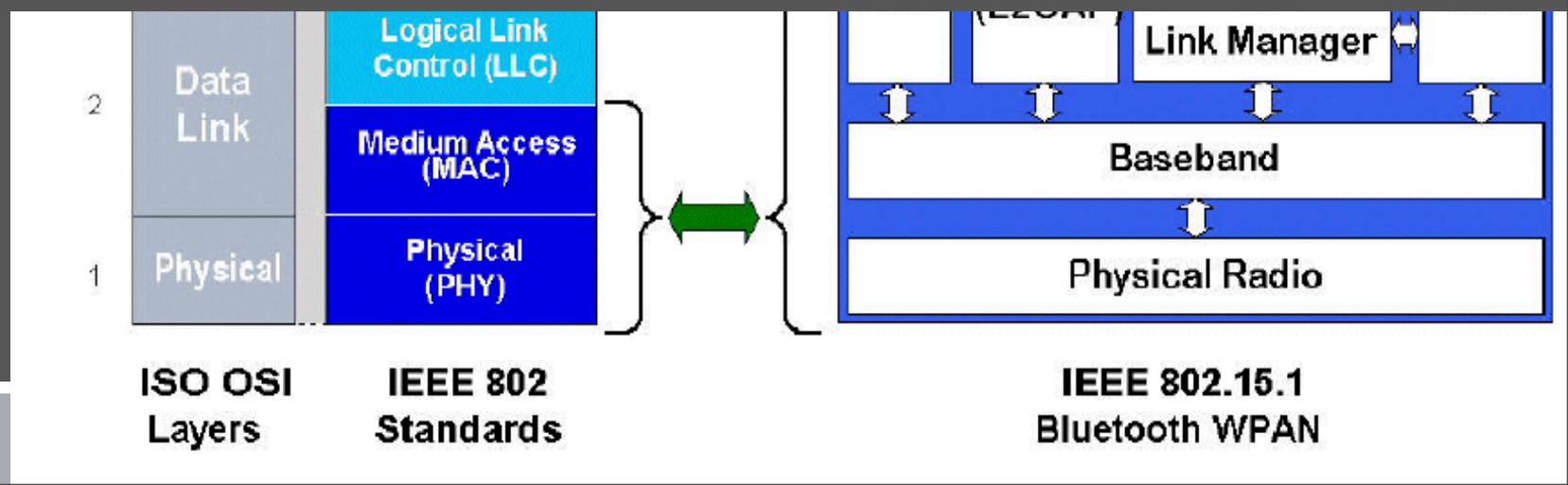
Bluetooth - Basisband

Das Bluetooth Basisband

- 0. Was ist das Basisband
- 1. Kanaldefinition
- 2. Geräteadressen
- 3. Bluetooth Paketformat
- 4. Physikalische Verbindung
- 5. Bluetooth Pakete
- 6. logische Kanäle

Bluetooth Basisbandschicht

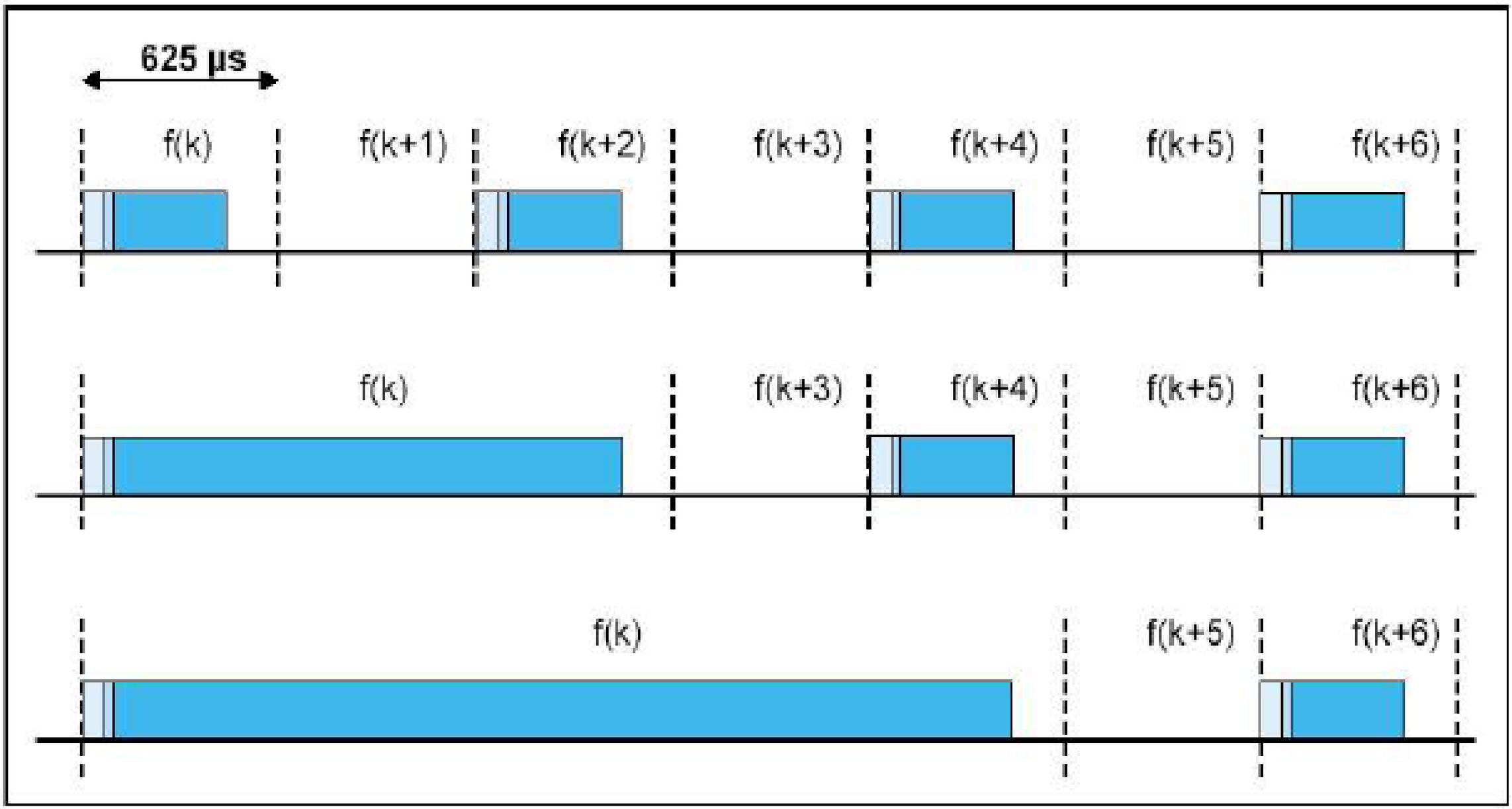
- Funktionen sehr vielfältig
- Ähnelt am meisten der Mac-Schicht aus dem 802 Modell
- Wandelt reinen Bitstrom in Rahmen
- Definiert physikalische Verbindungen
- Regelt den schnellen Frequenzwechsel



1. Kanaldefinition

- Ein Kanal ist dargestellt durch eine pseudo-zufällige Sprungsequenz zwischen den 79 möglichen Frequenzen im Bluetooth-Band.
- Gerät, das aktiv an einem Pikonetz teilnimmt, muss zur gleichen Zeit auf die gleiche Trägerfrequenz springen
- Pakete können nicht nur einen Zeitschlitz belegen, sondern auch 3 oder 5 (Multi-Slot-Pakete)
- Während eines Multi-Slot-Paketes findet kein Frequenzwechsel statt
- Nach der Übertragung findet der Frequenzwechsel laut Sprungfolge statt
- Sprungfolge durch Master vorgegeben

1. Kanaldefinition



2. Geräteadressen

- Jedem Bluetooth-Gerät ist eine weltweit eindeutige zugeordnet.
- Die Adresse ist vom IEEE 802-Standard abgeleitet.
- Die 48-Bit großen Adressen sind in drei Teile untergliedert:
  - LAP – Lower Address Part, mit 24 Bit
  - UAP – Upper Address Part, mit 8 Bit
  - NAP – Non-significant Address Part, mit 16 Bit.

Company Assigned	Company ID	
LAP	UAP	NAP

Die Felder LAP und UAP bilden den so genannten signifikanten Teil einer Bluetooth-Geräteadresse.

3. Bluetooth-Paketformat

Paket besteht aus 3 Teilen:

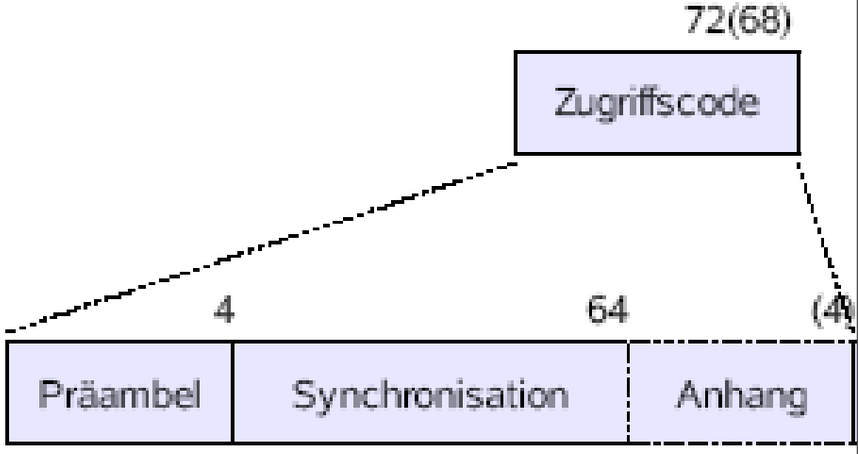
- Paket kann wie folgt zusammengesetzt werden:
  - Zugriffscode
  - Zugriffscode + Paketkopf
  - Zugriffscode + Paketkopf + Nutzdaten
- Zugriffscode mit 72 Bit falls Paketkopf folgt, sonst 68 Bit

3. Bluetooth-Paketformat-Zugriffscode

- Wozu:
  - Synchronisation und Identifikation eines Pikonetz
  - Geräteabfrage
  - Geräteruf

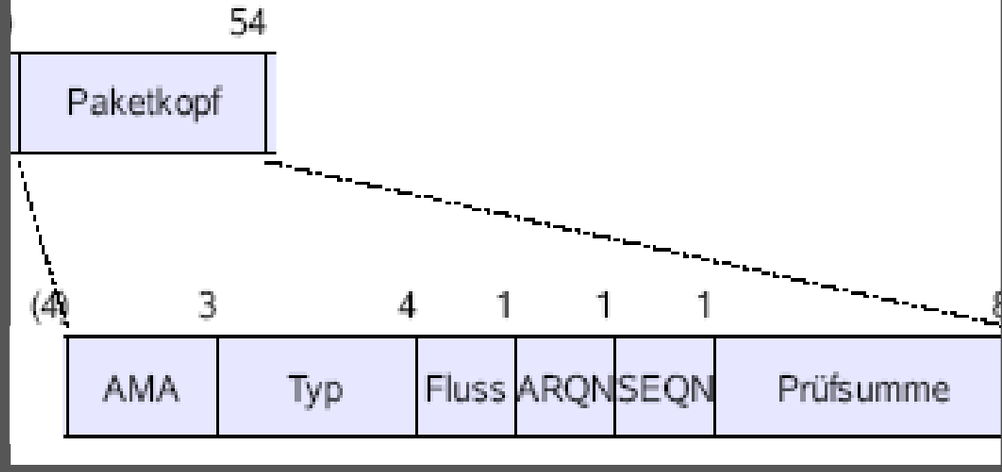
- 3 verschiedene Zugriffscodetypen:
  - Channel Access Code (CAC)
  - Device Access Code (DAC)
  - Inquiry Access Code (IAC)

- Komponenten des Zugriffscode:
  - Präambel (4 Bit)
  - Synchronisationsfeld (64 Bit)
  - Anhang (4 Bit) , falls Paketkopf folgt



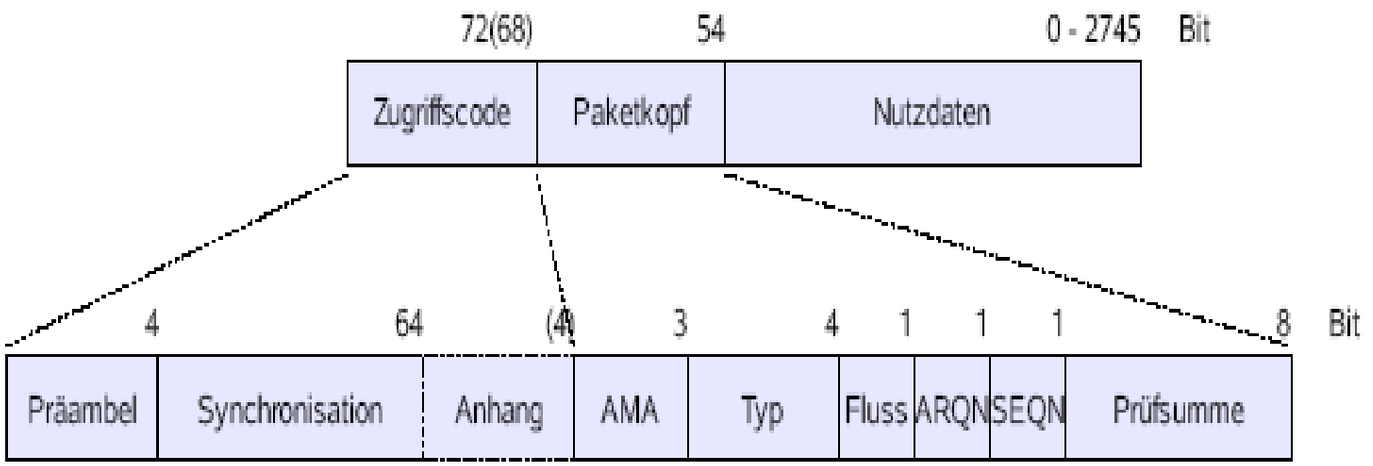
3. Bluetooth-Paketformat Paketkopf

- ist 54 Bit lang
- Enthält Informationen über die Verbindung
- Gesichert durch Vorwärtsfehlerkorrekturmechanismus
- eigentliche Headergröße 18 Bit
- 3-fache Übertragung 3x18 Bit = 54 Bit
- Der Header besteht aus 6 Feldern und umfasst die typischen Feldern des Layer 2 nach dem OSI-Modell
- Die Bestandteile:
  - Active Member Adress
  - Type (Typfeld)
  - Flow-Bits (Flusskontrolle)
  - ARQN (Bestätigungsnummer)
  - SEQN (Sequenznummer)
  - Prüfsumme



3. Bluetooth-Paketformat Nutzdaten

- Nutzdaten:
  - Bis zu 343 Bytes groß
  - Datenfeld besteht aus bis zu drei Segmenten
  - Nutzdatenheader
  - Eigentlichen Nutzdaten
  - eventuell Prüfsumme



4. Physikalische Verbindung

- Bluetooth bietet 2 verschieden Kommunikationstypen:
  - die leitungsvermittelte synchrone Kommunikation
  - die paketvermittelte asynchrone Kommunikation.
- In Bluetooth werden die beiden Vermittlungstechniken durch unterschiedliche physikalische Verbindungen realisiert:
  - Die synchrone verbindungsorientierte Kommunikation (SCO-Link)
  - Die asynchrone verbindungslose Kommunikation (ACL-Link)

4. Physikalische Verbindung

- Die synchrone verbindungsorientierte Kommunikation (SCO-Link)
  - symmetrische, leitungsvermittelte Punkt-zu-Punkt-Verbindung
  - Der Master reserviert in regelmäßigen Abständen Zeitschlitze
  - Master kann in einem festgelegten Zeitschlitz Daten an den Slave senden, der Slave kann in dem darauf folgenden Zeitschlitz seine Daten senden.
- Die asynchrone verbindungslose Kommunikation (ACL-Link) stellt einen verbindungslosen, paketvermittelnden Dienst zur Verfügung
  - ACL-Link wird genutzt, wenn Kanal nicht für SCO-Link reserviert ist.
  - Zwischen einem Master und einem Slave kann zu einem Zeitpunkt immer nur eine ACL-Verbindung aufgebaut sein
  - Rundruf des Master an alle Slaves pro Pikonetz möglich
  - Sicherungsmechanismen

5. Pakete Überblick

- Jeder Verbindungsart stehen verschieden Pakettypen zur Verfügung
- Verschiedene Pakettypen werden im Typfeld des Paketkopf mit 4 Bit kodiert
- Neben Paketen für ACL-Links und SCO-Links gibt es auch gemeinsame Pakete zur:
  - Abfrage von Slaves
  - Synchronisation der Sprungfolge
  - Bestätigung von Datenübertragungen.

5. Pakete

- Für SCO- und ACL-Links stehen jeweils 5 gemeinsame Pakettypen zur Verbindungssteuerung zur Verfügung:
  - Das ID-Paket
  - NULL-Paket
  - POLL-Paket
  - FHS-Paket
  - DM1-Paket

5. Pakete - SCO Link

- Für SCO-Links werden ausschließlich 1-Slot große Pakete verwendet.
- In Bluetooth sind drei HV-Pakete definiert. Die Pakete sind generell 240 Bit groß und unterscheiden sich in der Kodierung ihrer Nutzdaten
- HV-Pakete werden niemals erneut versendet und sind nicht über eine Prüfsumme geschützt.
- Neben den Paketen zur reinen Sprachübertragung steht noch ein 240 Bit großes Paket zur gemeinsamen Übertragung von Sprache und Nutzdaten zur Verfügung

- Die Pakete:
  - HV1-Pakete
  - HV2-Pakete
  - HV3-Pakete
  - DV-Pakete

5. Pakete - ACL Link

- Für ACL-Links können Pakete mit einer Länge von 1 Slot, 3 Slots oder 5 Slots genutzt werden.
- Hauptsächlich werden 3 Pakettypen unterschieden:
  - DMx-Pakete
  - DHx-Pakete
  - AUX1-Paket

5. Bluetooth Pakettypen

Segment	Paketcode	Slots	SCO-Link	ACL-Link
1	0000	1	NULL	NULL
	0001	1	POLL	POLL
	0010	1	FHS	FHS
	0011	1	DM1	DM1
2	0100	1	-	DH1
	0101	1	HV1	-
	0110	1	HV2	-
	0111	1	HV3	-
	1000	1	DV	-
	1001	1	-	AUX1
3	1010	3	-	DM3
	1011	3	-	DH3
	1100	3	-	-
	1101	3	-	-
4	1110	5	-	DM5
	1111	5	-	DH5

5. Bluetooth Datenpakete

							Asymm. Datenrate [kbps]	
	Type	Header [Byte]	Nutzlast [Byte]	FEC	CRC	Symm. Datenrate [kbps]	Hinkanal	Rückkanal
Link-Pakete	ID	-	-	-	-	-	-	-
	NULL	-	-	-	-	-	-	-
	POLL	-	-	-	-	-	-	-
	FHS	-	18	2/3	ja	-	-	-
ACL-Pakete	DM1	1	0-17	2/3	ja	108,8	108,8	108,8
	DH1	1	0-27	nein	ja	172,8	172,8	172,8
	DM3	2	0-121	2/3	ja	258,1	387,2	54,4
	DH3	2	0-183	nein	ja	390,4	585,6	86,4
	DM5	2	0-224	2/3	ja	286,7	477,8	36,3
	DH5	2	0-339	nein	ja	433,9	723,2	57,6
	AUX1	1	0-29	nein	nein	185,6	185,6	185,6
SCO-Pakete	HV1	-	10	1/3	nein	64,0	-	-
	HV2	-	20	2/3	nein	64,0	-	-
	HV3	-	30	nein	nein	64,0	-	-
	DV	1	10+(0-9) D	2/3 D	ja D	64,0+57,6 D	-	-

6. Logische Kanäle

- Beziehen sich auf verschieden Typen von Kanälen, die über eine physikalische Verbindung laufen
- Daten, die über die physikalische Verbindung übertragen werden haben unterschiedliche logische Bedeutung
- Bluetooth kennt 5 verschiedene Kanäle für Steuer- und Benutzerinformation

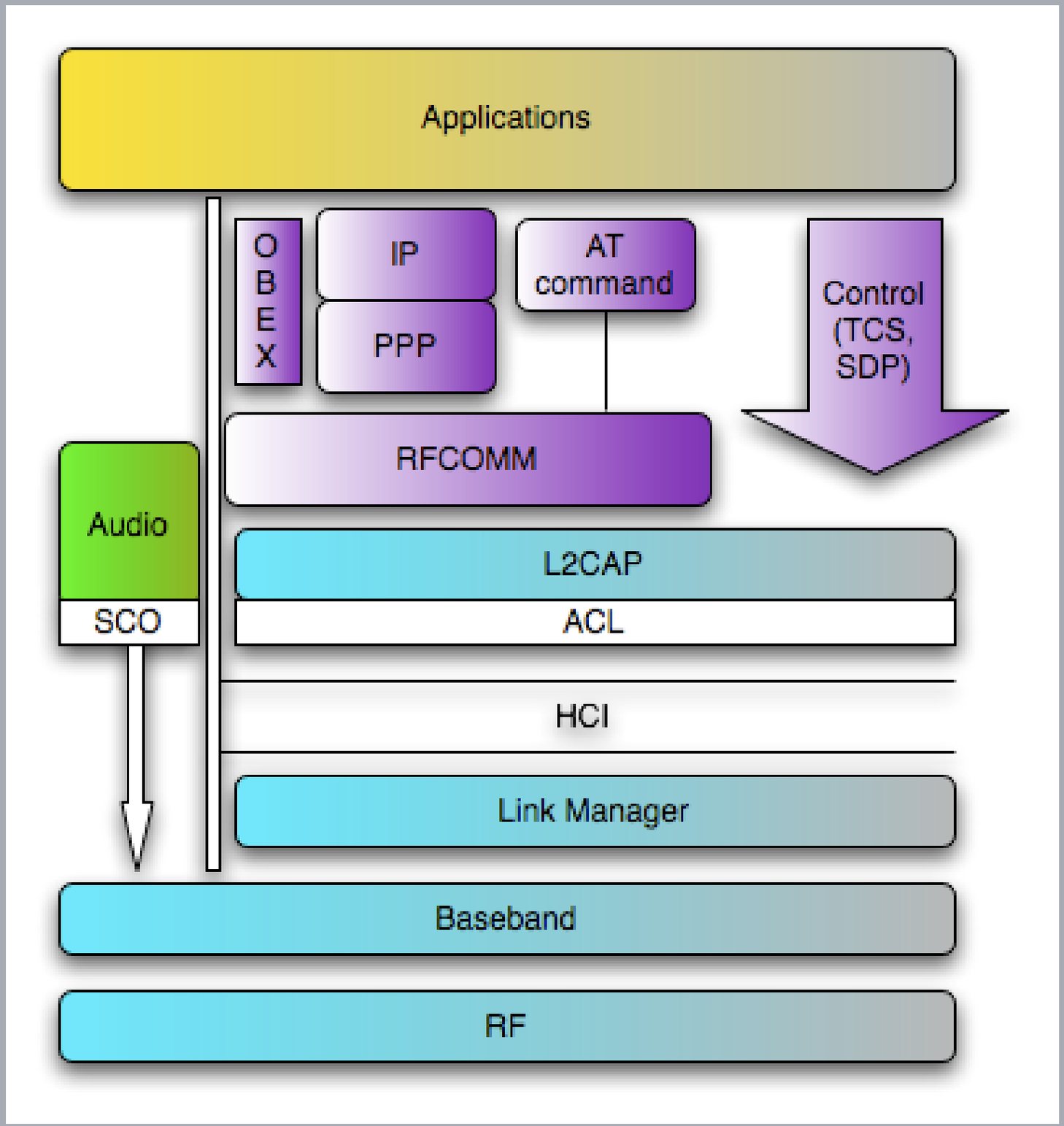
6. Logische Kanäle

- Link Control (LC)
  - Stellt einen Steuerkanal dar, der auf der Ebene der Verbindungssteuerung verwendet wird.
- Link Manager (LM)
  - Stellt Steuerkanal dar, der auf der Ebene des Link-Managers verwendet wird.
- User Asynchronous (UA)
  - logischer Kanal auf der Basis der physikalischen ACL-Links
- User Isochronous (UI)
  - logischer Kanal auf der Basis der physikalischen ACL-Links
- User Synchronous (US)
  - synchronen logischen Kanal auf Basis des SCO-Links
  - Die Nutzerkanäle dienen der Übertragung von asynchronen, isochronen und synchronen Daten,
  - die aus der Benutzerebene, den Applikationen kommen.

Link Manager Protocol

LMP

Im Protocol Stack



LMP

Aufgaben des Link Manager

- Verbindungsaufbau + Verbindungsabbau
- Sicherheit
  - Authentifizierung
  - Authorisierung
  - Verschlüsselung
- Power Control Prozeduren

Verbindungsaufbau

- 3 Phasen:
  - 1. Inquiry
  - 2. Pairing
  - 3. Paging

Discoverable modes

- Discoverable: Antwortet auf alle Inquiries
- Non Discoverable: Antwortet nicht auf Inquiries
- Limited Discoverable: Antwortet nur bestimmte Zeit lang auf Inquiries

Inquiry

- Vorgegebene „Hopping Sequence“
- Übergabe von Paging Parametern (MAC Addr, Clock Offset...)

Paging

- Voraussetzung: verbindungsbereites Zielgerät (--> im Page-Scan-Modus)
- Frequency Hopping Synchronisation
- Slaves synchronisieren sich auf den Master

Paging - Initialisierung

- Slaves springen 1 mal/s
- Master springt 3200 mal/s

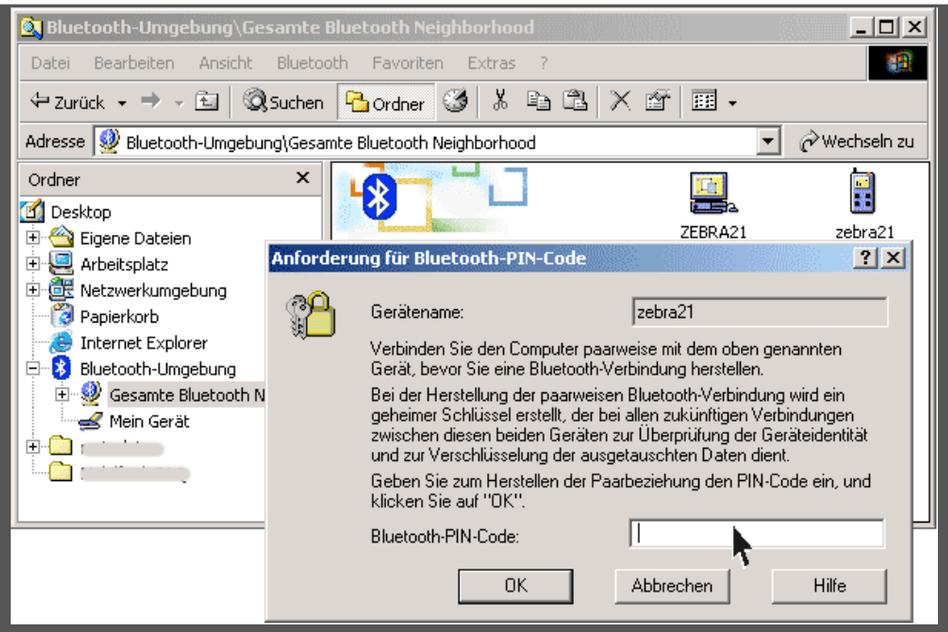
Paging - nach der Initialisierung

- Slave und Master springen 1600 mal/s
- Sprungsequenz auf Grund der BD_ADDR + Clock Offset des Masters

Sicherheit

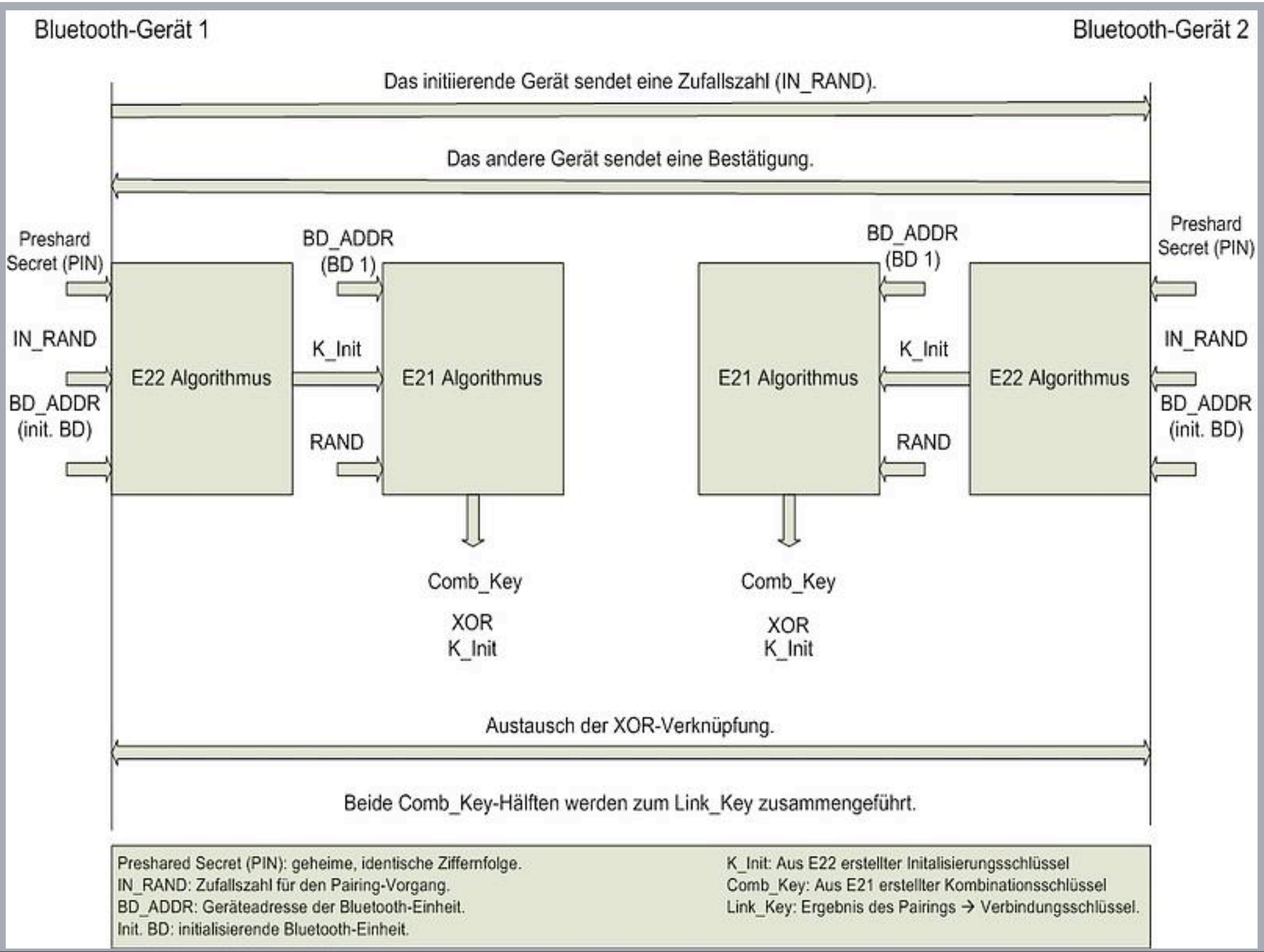
- 3 Modi:
  - Modus 1: Non-secure mode
  - Modus 2: Service level enforced security
  - Modus 3: Link level enforced security

Sicherheit - Pairing



LMP

Sicherheit - Pairing

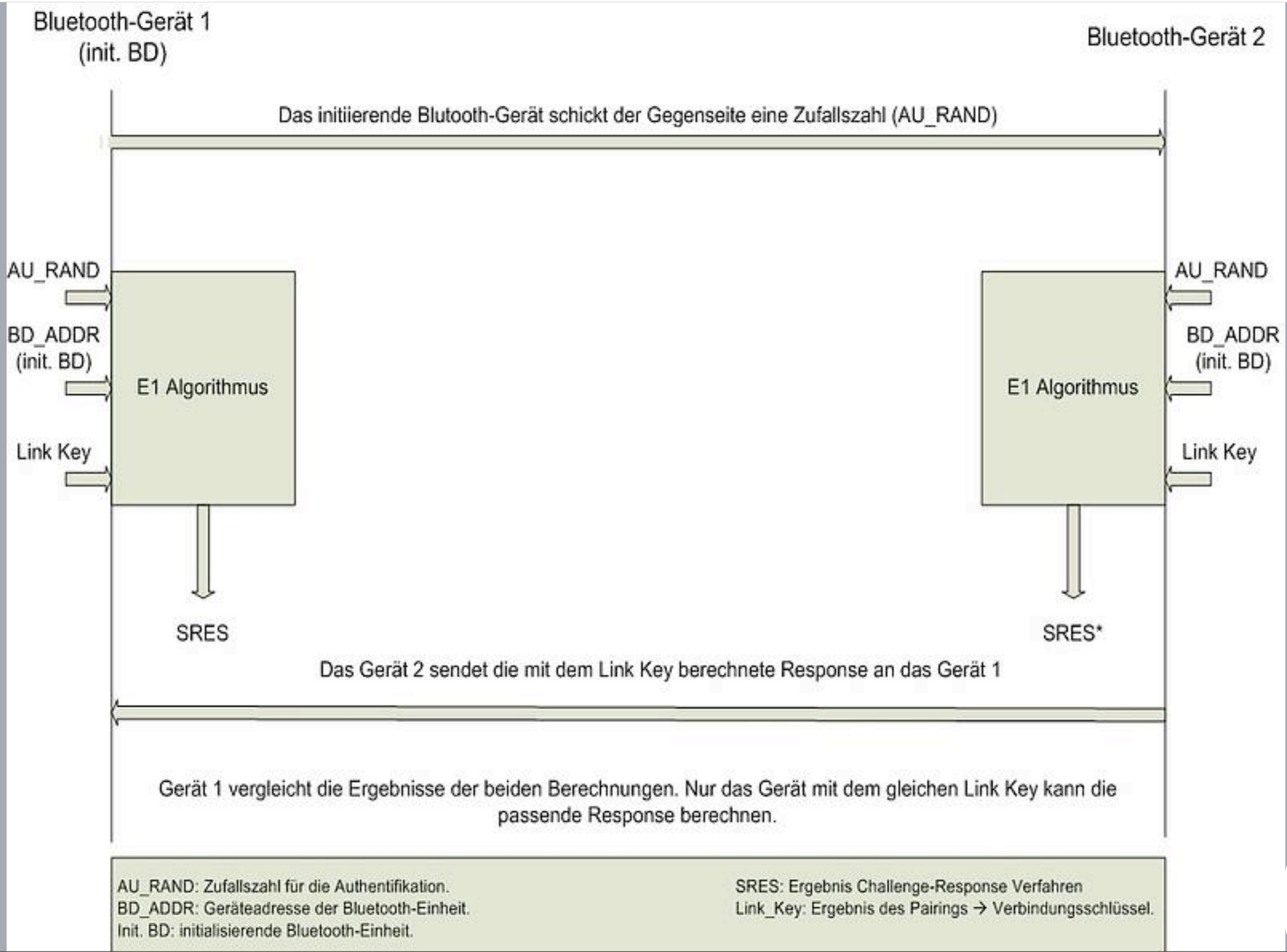


LMP

Sicherheit - Authentification

LMP

Sicherheit - Authentification

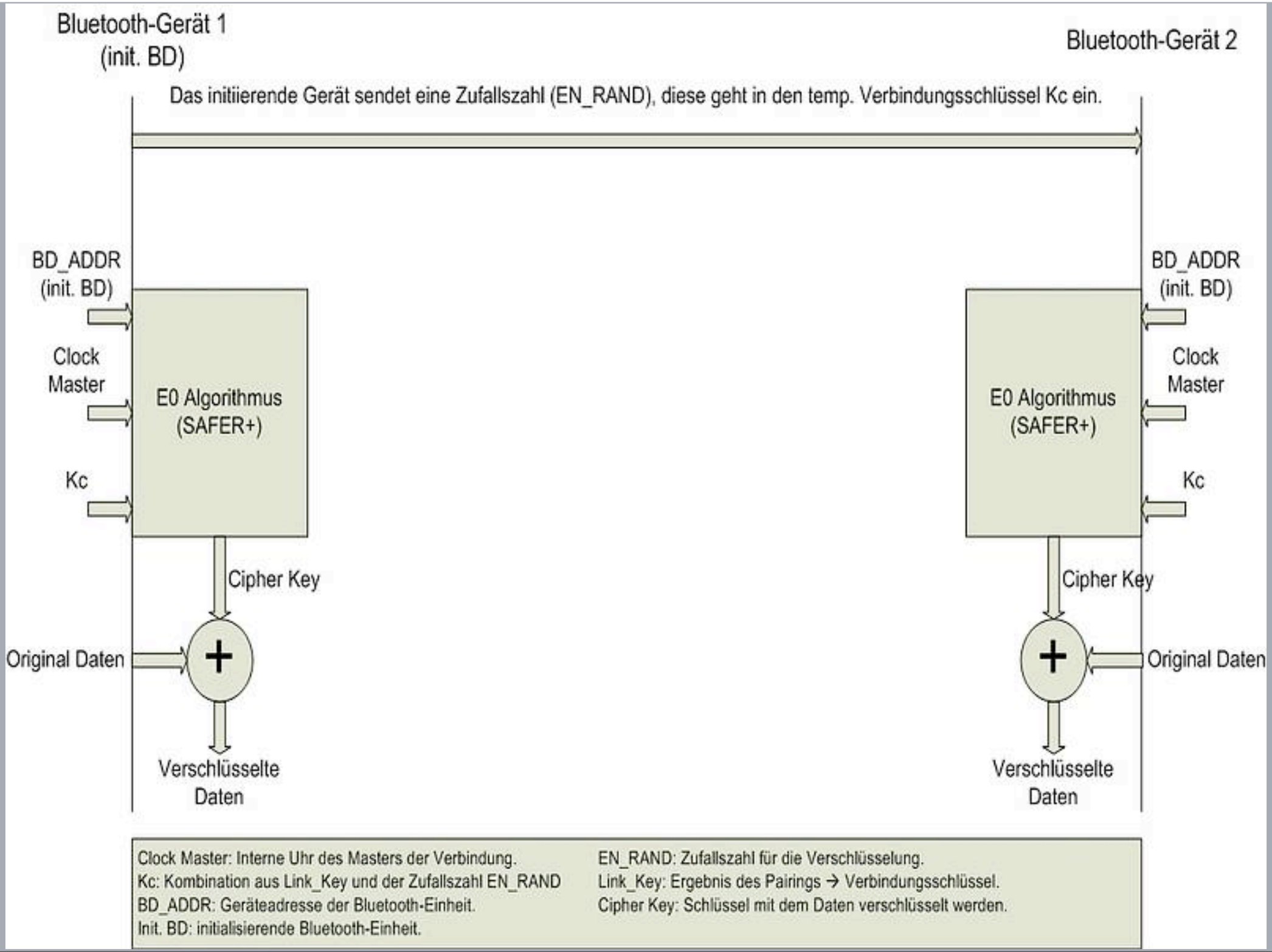


LMP

Sicherheit - Verschlüsselung

LMP

Sicherheit - Verschlüsselung



LMP

Energiesparmodi

- Hold
- Sniff
- Park

LMP

Energiesparmodi: Hold

- Hold:
  - Erst nach Ablauf einer Zeitspanne wird der Slave wieder kontaktierbar
  - Gerät bleibt aktiv im Piconet (AM_ADDR)
  - Keine ACL Connections mehr
  - SCO Connections bleiben erhalten
  - Timer läuft weiter

Energiesparmodi: Sniff

- Sniff:
  - Gerät bleibt aktiv im Piconet (AM_ADDR)
  - Keinerlei Connections mehr
  - Timer läuft weiter
  - Hört in festgelegten Abständen ins Netz

Energiesparmodi: Park

- Park:
  - Gerät nichmehr aktiv im Piconet (PM_ADDR)
    - PM_ADDR = Master kann Slave wecken
  - Gerät erhält AR_ADDR (Active Request Addr.)
    - AR_ADDR = Slave kann „entparken“ fordern
  - Timer läuft weiter

Power Control

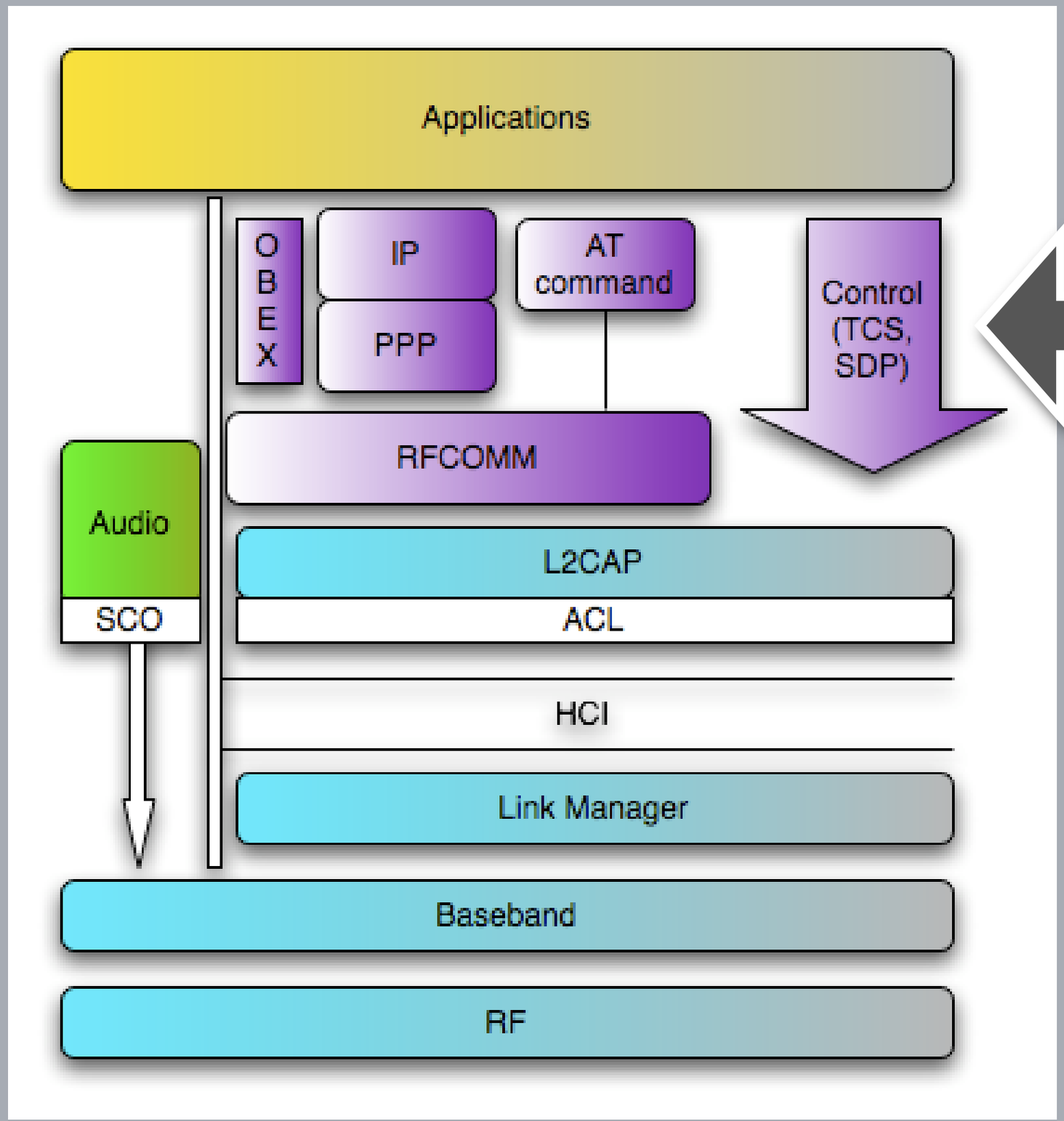
- Regelung pro Gerät
- Pakettypen (Implementierung optional)
  - LMP_incr_power_req,LMP_decr_power_req
  - LMP_max_power,LMP_min_power
- Wenn Empfang zu schlecht --> FEC an (LMP_preferred_rate)

Service Discovery Protocol

SDP

SDP

Im Protocol Stack



SDP

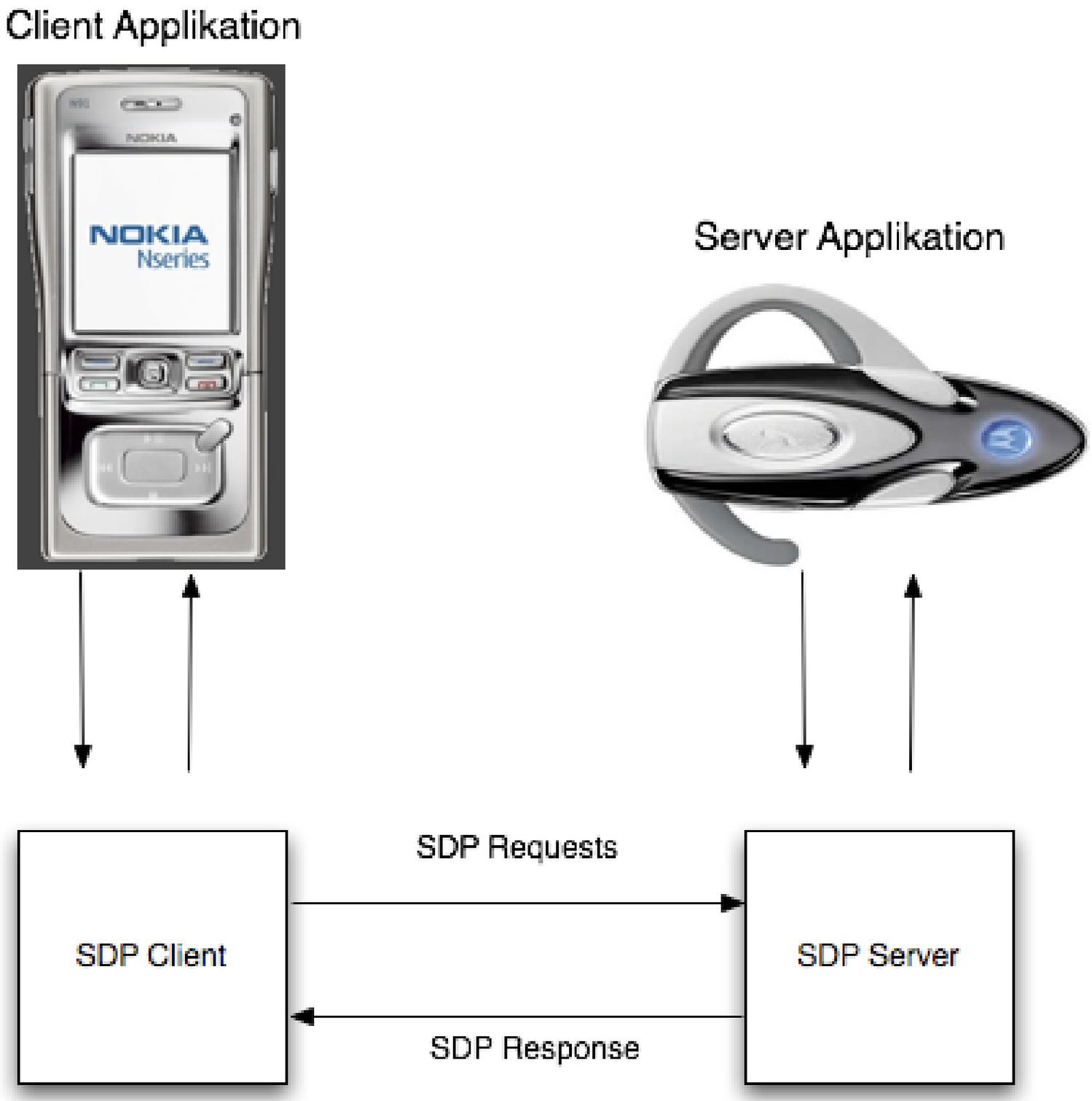
Motivation für SDP

- Statische Netze: Verzeichnisdienst (LDAP)
- Dynamisches Bluetooth Netz: ?

Aufgaben des SDP

- Dienste auf anderen Geräten zu finden
- Auskunft über eigene Dienste geben
- NICHT: Gefundene Dienste nutzen

Funktionsweise des SDP



SDP PDU

- PDU ID
- Transaction ID (request<-->response)
- Parameter Length

Service Record: Service Attributes

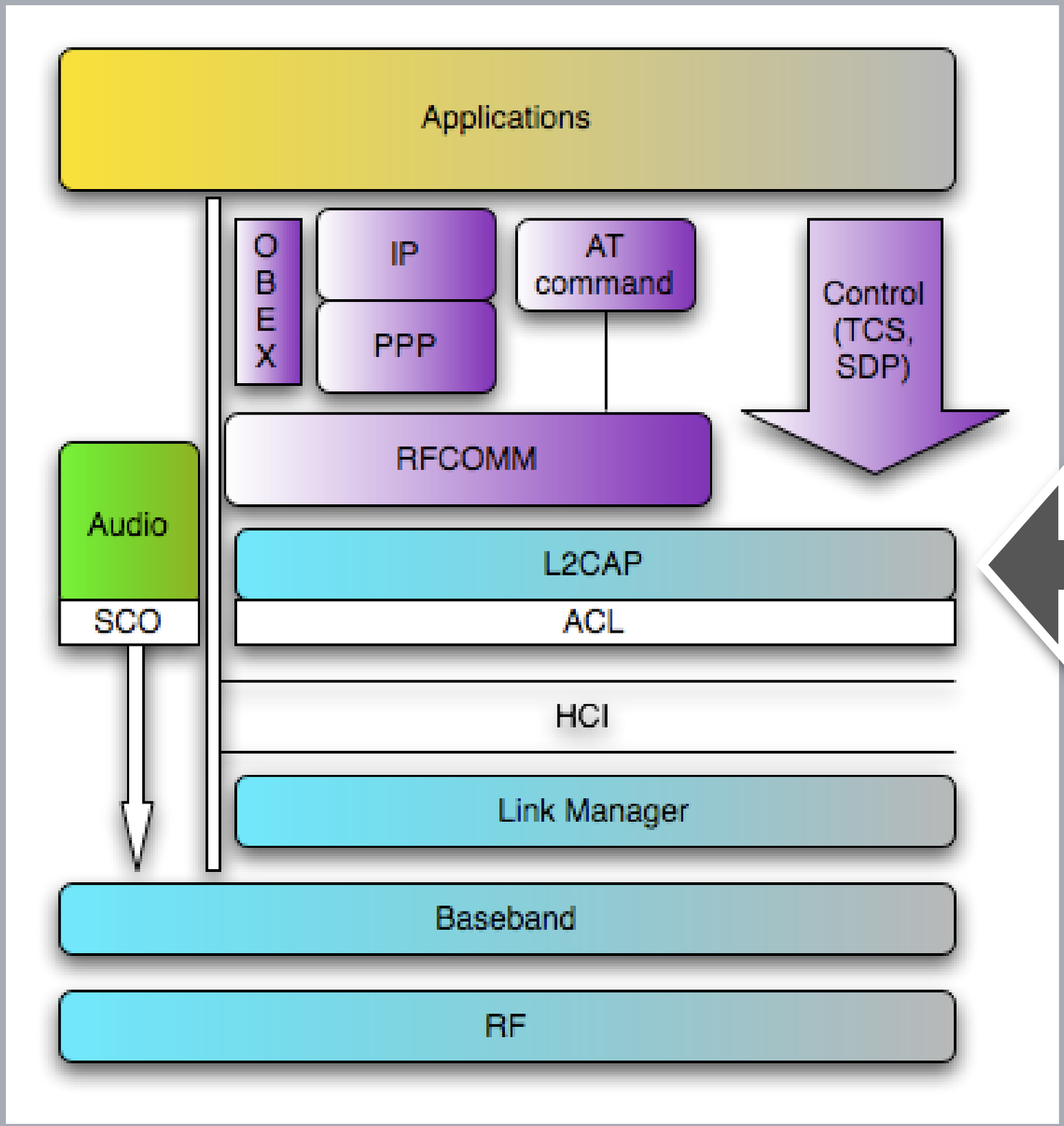
- ServiceID (eindeutig)
- IconURL
- DocumentationURL
- ClientExecutibleURL
- Service Description
- ServiceRecordHandle
- ServiceClassIDList
- ServiceRecordState
- Service Name
- ProtocolDescriptionList
- BrowseGroupList
- LanguageBaseAttributeIDList
- ServiceInfoTimeToLive
- ServiceAvaliability
- BluetoothProfileDescriptorList

Logical Link Control and Adaptation Protocol

L2CAP

L2CAP

Im Protocol Stack



L2CAP

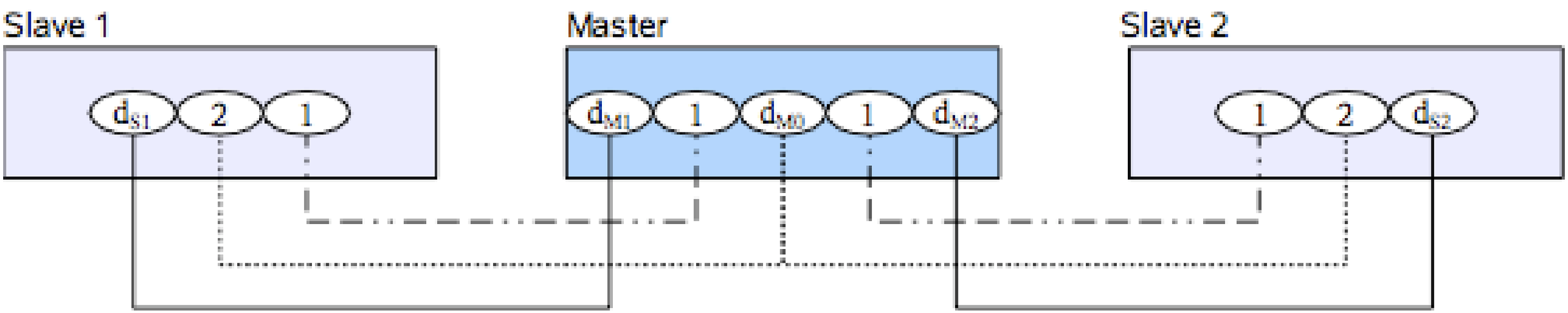
Aufgaben von L2CAP

- NUR ACL! Kein SCO (in aktueller Spezifikation)
- Logische Kanäle bereitstellen (eindeutige CID)
- Fragmentierung und Reassemblierung
- Multiplexing von Protokollen+Datenströmen (z.B. mehrere Paketquellen)
- Gruppenverwaltung
- QoS (z.B. maximale Größe der Nutzdaten)

Typen virtueller Kanäle

- Signalisierung
  - Zwischen L2CAP Instanzen, Sicherung der Kommunikation -> bevorzugt. CID: 1
- Verbindungslos
  - Punkt->Multipunkt: Broadcast. Senden: Dynamische CID / Empfangen: CID 2
- Verbindungsorientiert
  - Punkt->Punkt, Dynamische CID >=64

Vorhandene Kanäle: Beispiel

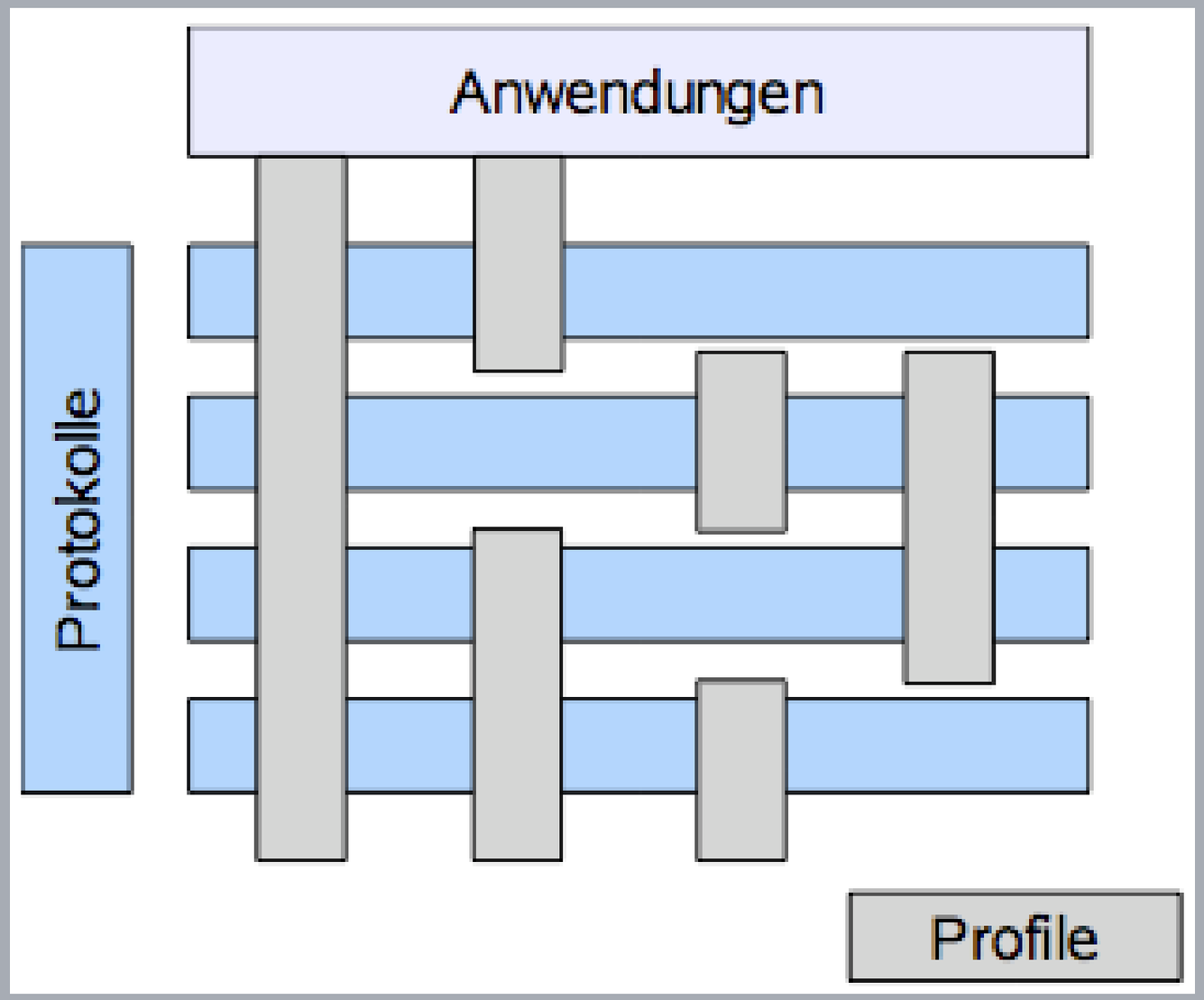


Services

- Connection: setup , configure , disconnect
- Data: read , write
- Group: create, close, add member, remove member , get membership
- Information: ping, get info, request a call-back at the occurrence of an event
- Connection-less Traffic: enable, disable

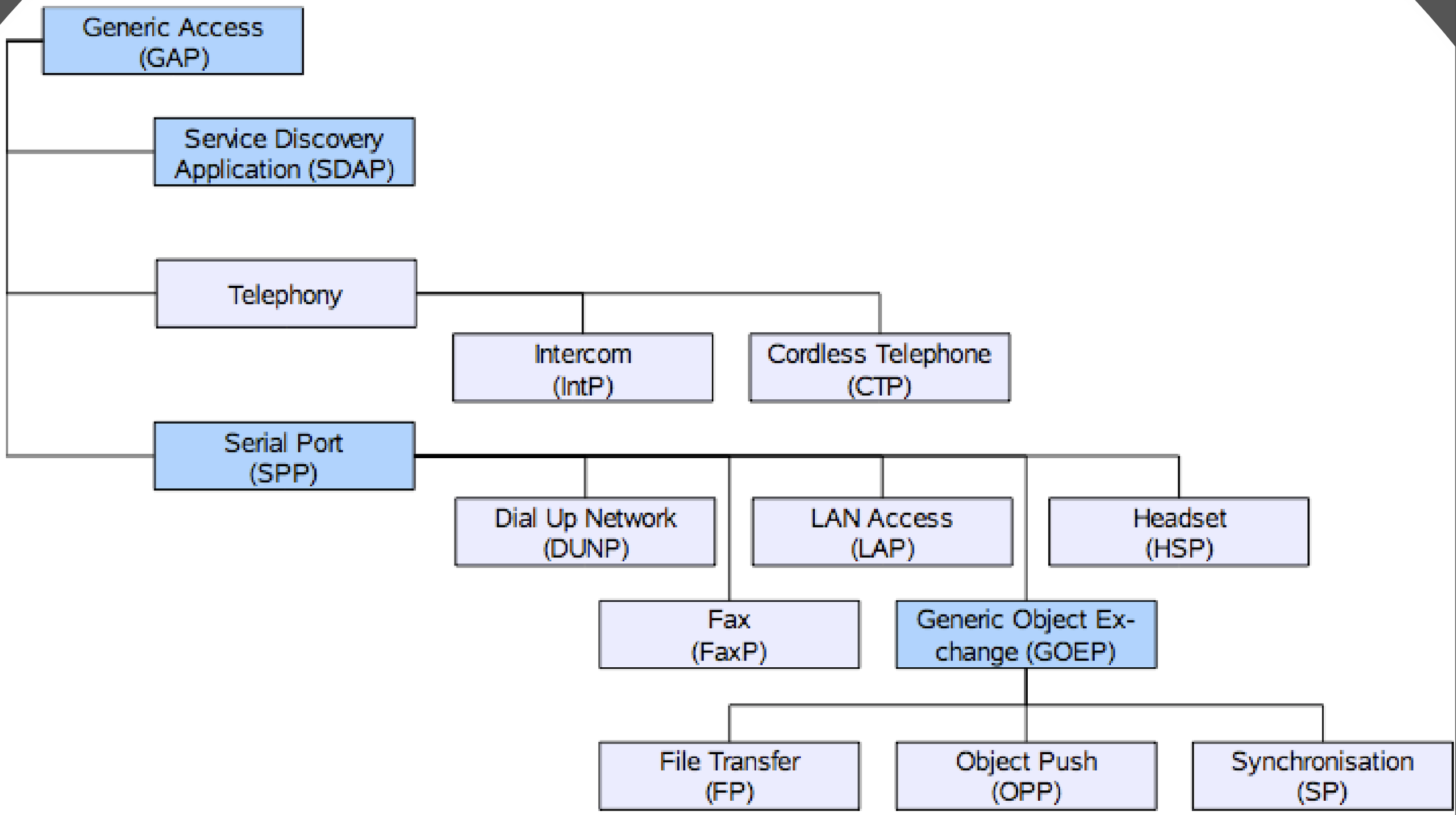
Bluetooth Profile

Was sind Profile?



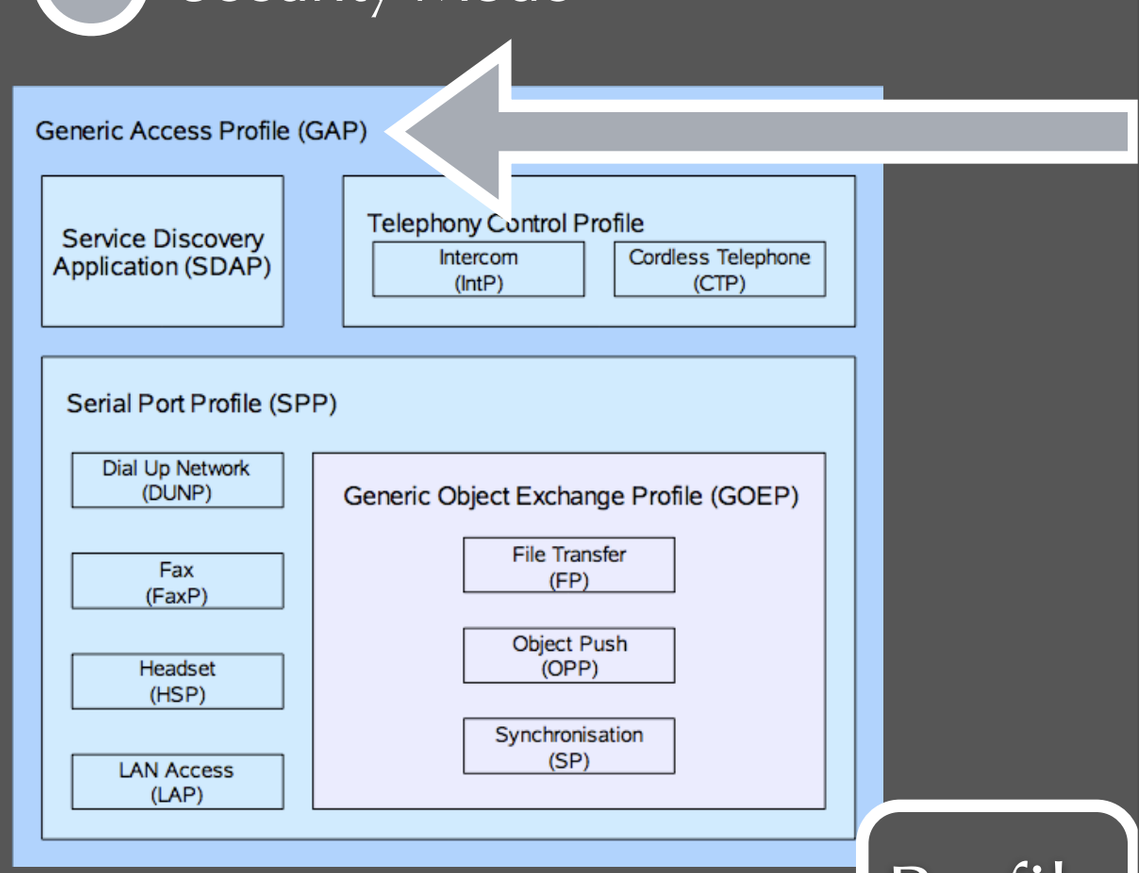
„Ein Profil beschreibt, welche Protokolle und welche Parameter gesetzt werden müssen, damit ein Bluetooth-Gerät ein bestimmtes Nutzungsmodell erfüllt.“

Welche Profile gibt es?



Generic Access Profil (GAP)

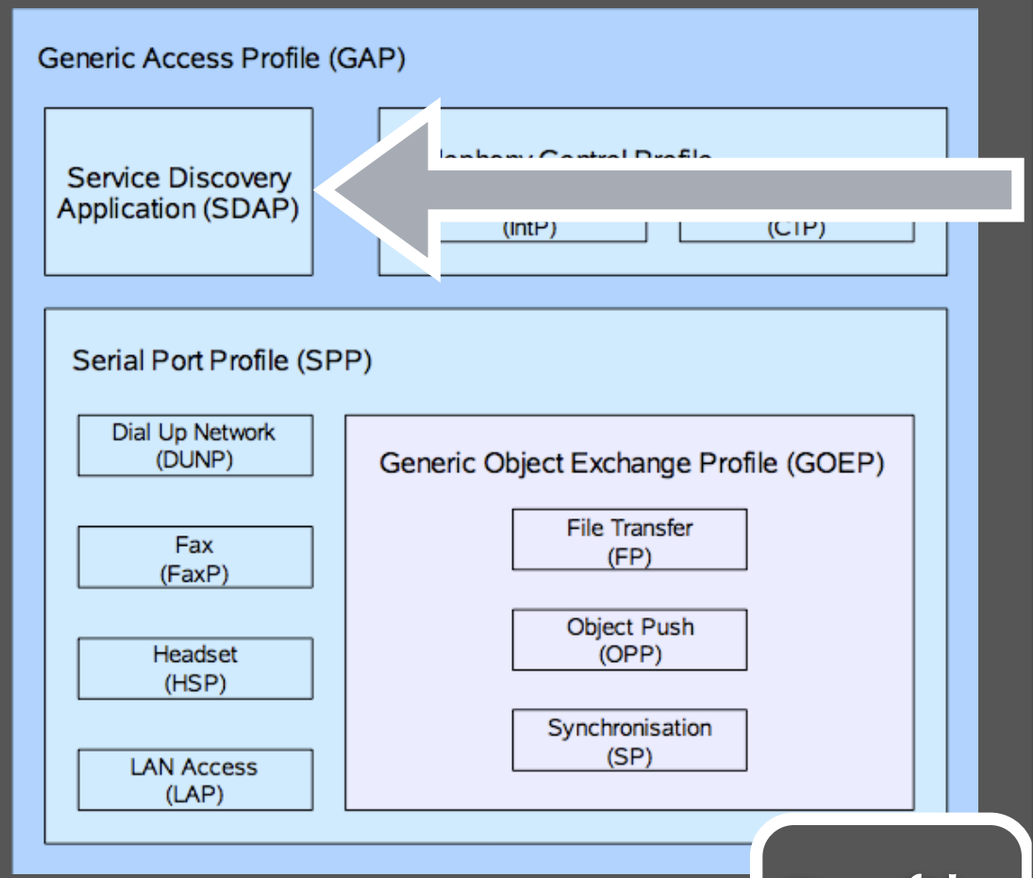
- Verbindungsaufbau und -verwaltung:
  - Link Establishment
  - Channel Establishment
  - Connection Establishment
- Betriebsmodi:
  - Discoverable Mode
  - Connectability Mode
  - Pairing Mode
- Sicherheit
  - Authentification
  - Security Mode



Profile

Service Discovery Application Profil (SDAP)

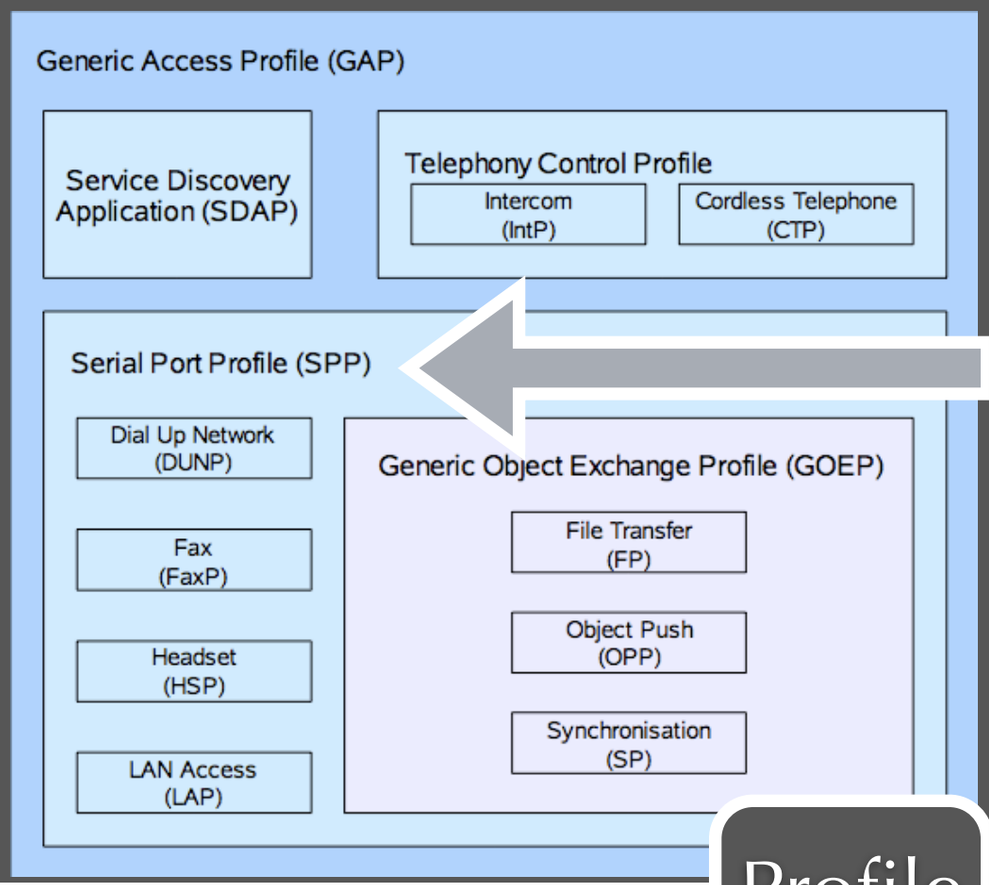
- Zugriffsschnittstelle auf SDP
- SDAP baut auf GAP auf



Profile

Serial Port Profile (SPP)

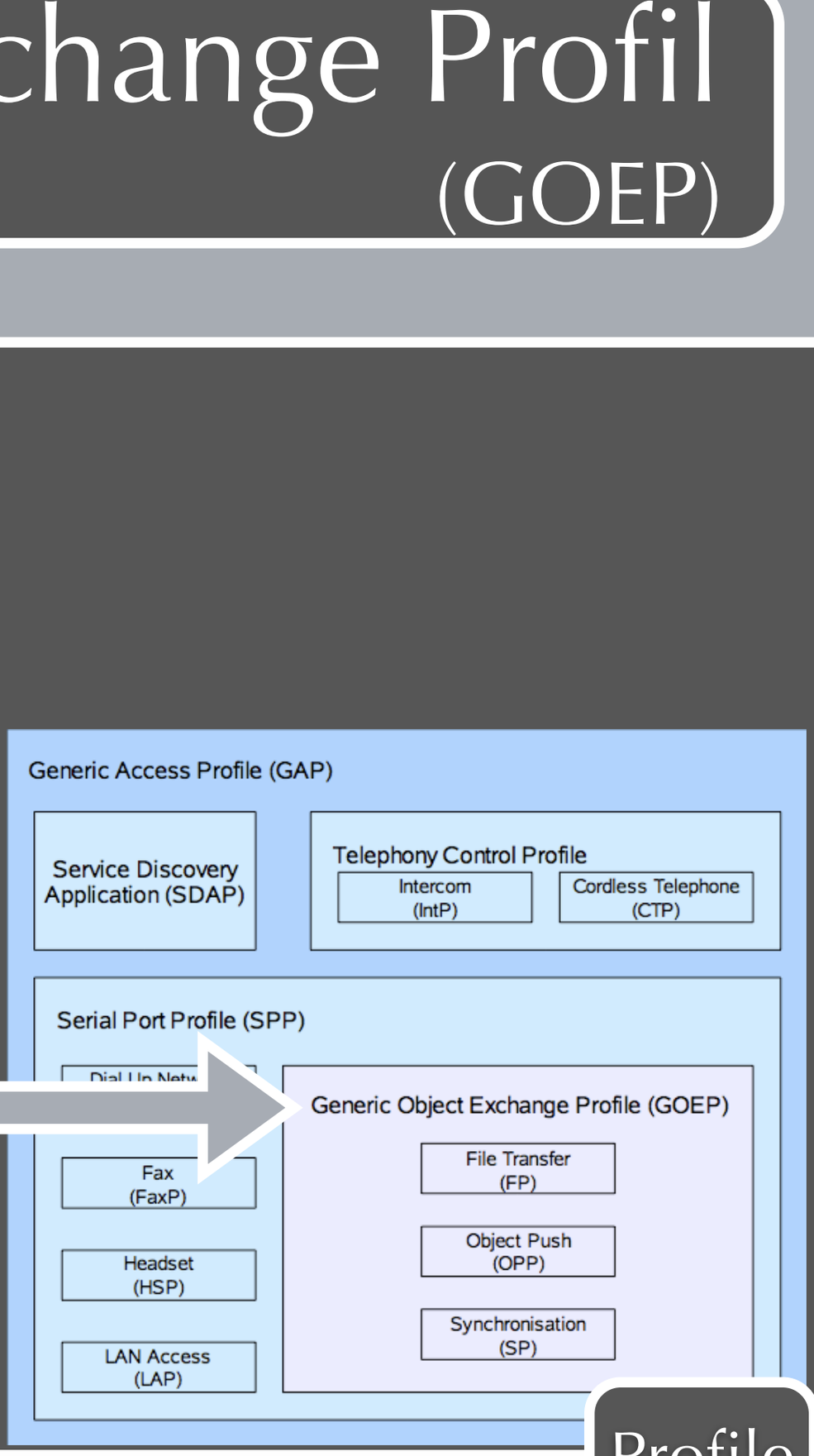
- Kabelersatz
- Baut auf GAP auf
- Nutzt RFCOMM Protokoll



Profile

Generic Object Exchange Profil (GOEP)

- Client <-> Server Beziehung
- Austausch komplexer Objekte
- Nutzt RFCOMM Protokoll



Profile

Übersicht über Profile

Profil-Name	Beschreibung	Beispielgeräte (Auswahl)
Generic Access Profile (GAP)	Verbindungsaufbau und -steuerung	-
Service Discovery Application Profile (SDAP)	Anbieten und Erkennen von Diensten	-
Serial Port Profile (SPP)	Ersatz serieller Datenverbindungen	PC, Notebook, PDA, Handy, Drucker, Modem
Generic Object Exchange Profile (GOEP)	Allgemeiner Objektaustausch, definiert eine Client/Server-Beziehung	-
Dial Up Network Profile (DUNP)	Einwahlzugang über Modem oder Handy	Analog-Modem, ISDN-Modem, PC, Notebook, PDA, Handy
Fax Profile (FaxP)	Steuerung von Faxdiensten zwischen Geräten	Analog-Modem, ISDN-Modem, PC, Faxgerät
Headet Profile (HSP)	Steuerung von Headsets/Freisprecheinrichtungen	Headset, Handy, PC, PDA
LAN Access Profile (LAP)	LAN-Zugriff üder das Point-to-Point-Protokoll (PPP)	ISDN-,DSL-,LAN-Access-Point
Cordless Telephony Profile (CTP)	Unterstützung für schnurlose Telefondienste (Funktionalität einer DECT[23]-Umgebung)	Handy, Basisstation
Intercom Profile (IntP)	Kommunikation zwischen Handsets (Terminals, eine Art Walkie-Talkie	Handy, PDA, Gegensprechanlage
File Transfer Profile (FP)	Übertragung von Dateien zwischen Bluetooth-Geräten	Handy, PC, PDA, Notebook
Object Push Profile (OPP)	Übertragung von Datenobjekten (z.B. Austausch von Visitenkarten – vCard – und Terminen – vCal)	PC, PDA. Notebook, Handy, Drucker, Scanner, Faxgerät
Synchronisation (SP)	Synchronisation von Geräten auf Basis von typischen PIM-Daten (Personal Information Manager)	PC, Notebook, Handy, PDA

Profile

Quellen

- 22C3-536-en-bluetooth_hacking.m4v
- 23C3-1733-en-bluetooth_hacking_revisited.m4v
- Heiko Holtkamp - Einführung in Bluetooth
- Dr.-Ing. Dipl.-Inform. Bernhard Löhlein - Bluetooth–Sicherheitsarchitektur und Angriffspunkte
- Die Datenschleuder #88 - Inside Bluetooth Security
- http://www.palowireless.com/
- http://www.chaostal.de/cgi-bin/parser.cgi?input=article/bluetooth
- Sandra Hagen, Christian Wöck - Bluetooth

Hauptseminar „Dienste & Infrastrukturen mobiler Systeme“
Wintersemester 03/04
Institut für Informatik
Ludwig Maximilians Universität München

- www.holtmann.org/papers/bluetooth/saimba_slides.pdf
- Eugene A. Gryazin - Service Discovery in Bluetooth

LMP + SDP + L2CAP + Profile

Java APIs for Bluetooth Wireless Technology (JABWT)

Stephan Helten

(sh094@hdm-stuttgart.de)

Gliederung des Vortrags

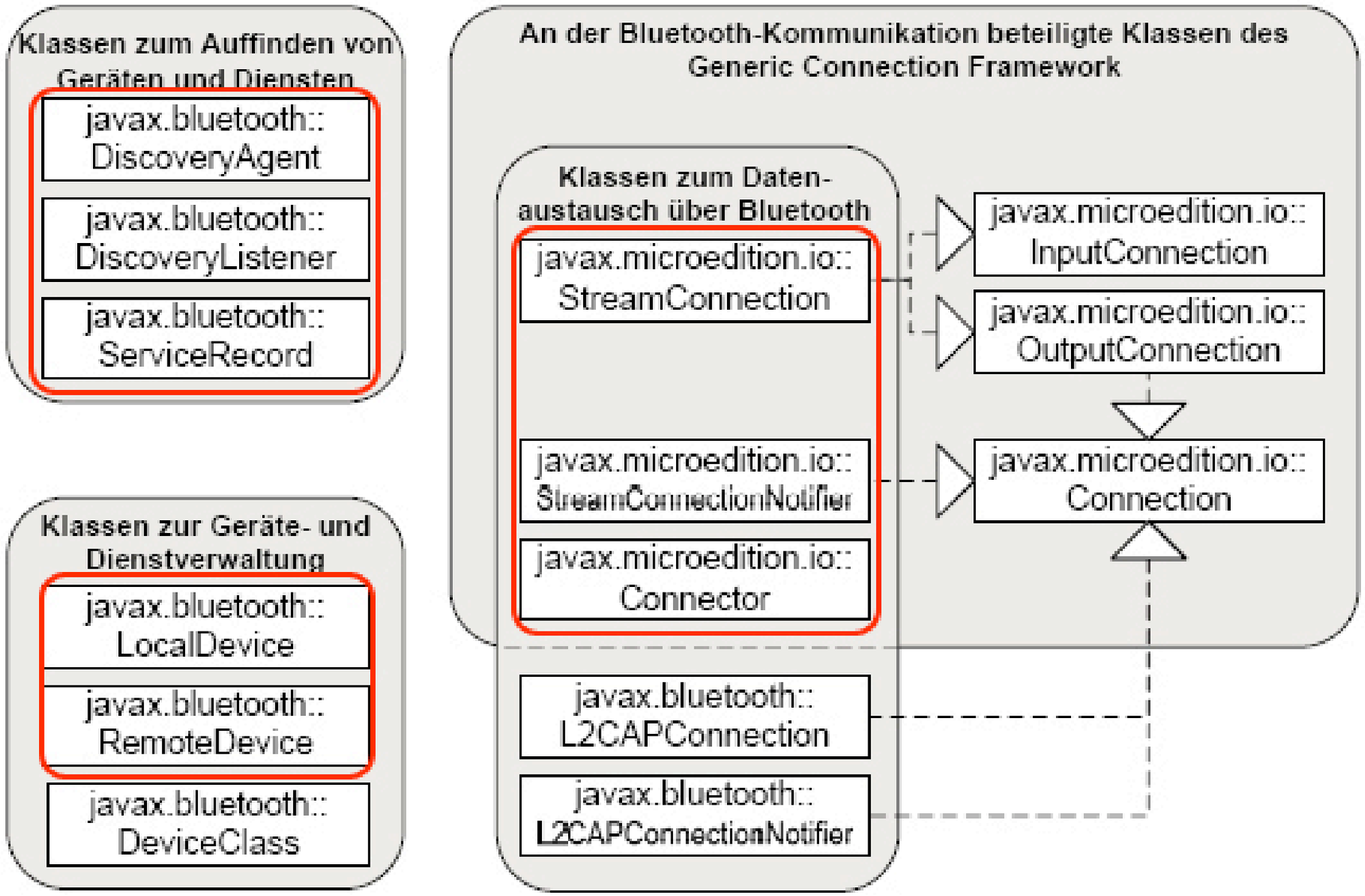
1. **Aufbau** und Einordnung der APIs
2. Phasen der **Bluetooth-Kommunikation** einer Anwendung
3. Die **Verwendung** der APIs

Aufbau und Einordnung der JABWT

Aufbau und Einordnung der JABWT

- Klassen zur Geräte- und Dienste**verwaltung**
- Klassen zur Geräte- und Dienste**suche**
- Klassen zur eigentlichen Datenkommunikation

Aufbau und Einordnung der JABWT



Phasen der Bluetooth-Kommunikation einer Anwendung

Phasen der Bluetooth-Kommunikation einer Anwendung

1. Geräte konfigurieren / verwalten
2. Dienste verwalten (Server)
3. Gerätesuche
4. Dienstsuche (Client)
5. Erst jetzt: Eigentlicher Datenaustausch

Die Verwendung der JABWT

1.Geräte verwalten | 2.Dienste verwalten | 3.Gerätesuche | 4.Dienstsuche | 5.Datenaustausch

1.Geräte konfigurieren / verwalten

Konfiguration

- Voraussetzung: Korrekte Konfiguration des Bluetooth-Gerätes
- Benutzerschnittstelle im Handy: Bluetooth Control Center (BCC)
- Möglichkeiten der Java ME zur Konfiguration beschränkt

Klasse: LocalDevice

- (Singleton-) **Klasse LocalDevice** repräsentiert das lokale Gerät
- Instanz wird von Methode **getLocalDevice ()** zurück gegeben
- Verwendung: Konfigurationsparameter **lesen** und **ändern**.

Klasse: LocalDevice

- Wichtige Methoden zum **Lesen** von Konfigurationsparametern:
  - **getBluetoothAddress()**
  - **getFriendlyName()**
  - getDeviceClass()
    - Gibt *Class of Device* des Gerätes zurück: Funktionen bestimmen Geräteklassen (Computer, Imaging, Phone,…)
  - getDiscoverable()

Klasse: LocalDevice

- Einzige Methode zum **Ändern** von Konfigurationsparametern:
  - setDiscoverable(int DiscoveryAgent.MODE)
  - Mode-Konstanten in DiscoveryAgent:
    - NOT_DISCOVERABLE (nicht auffindbar)
    - GIAC (dauerhaft auffindbar und kontaktierbar)
    - LIAC (auffindbar für eine bestimmte Zeit)

Programmcode: LocalDevice

```
//Instanz von LocalDevice holen
LocalDevice dev = LocalDevice.getLocalDevice();

//Gerät auffindbar machen
try {
   dev.setDiscoverable(DiscoveryAgent.GIAC);
} catch (BluetoothStateException e)
{
    //Änderung nicht übernommen
    //z.B. wenn BCC dies verweigert
}

//Gerätenamen auslesen
String name = dev.getFriendlyName();
//Geräteadresse auslesen
String address = dev.getBluetoothAddress();
//Discoverable-Mode auslesen
int discoverable = dev.getDiscoverable();
```

Klasse: RemoteDevice

- Repräsentiert Remote-Geräte
- Ähnlich aufgebaut wie LocalDevice
  - Ebenfalls die Methoden:
    - getBluetoothAddress()
    - getFriendlyName()
- Instanz holen über:
  - **getRemoteDevice(Connection con)**

2.Dienste verwalten

Dienste

- Nur über einen Dienst kann eine Verbindung hergestellt werden
- Anwendung kann Dienst **bereitstellen** (Server) oder **nutzen** (Client)
- Dienst muss **veröffentlicht** und **aufgefunden** werden
- Gerät (Client) kann zu bestehendem Dienst eines anderen Geräts (Server) verbinden

Service Discovery Database

- Registrierung der Dienste in der ***Service Discovery Database*** (SDDB)
  - Jedes Gerät hat eigene SDDB
- SDDB besteht aus Einträgen von ***Service Records***
- Ein Service Record repräsentiert einen Dienst

Service Record

- besteht aus **Service-Attributen**
  - Ein Service Attribut besteht aus
    - einem **ID** und **value** –Paar
      - ID: int
      - value: DataElement
  - beschreiben den Dienst
  - Wichtigstes Service Attribut:
    - ServiceID:
      universally unique identifier (UUID-Klasse)

Bereitstellung eines Dienstes

Server stellt einen Dienst in drei Schritten:

1. **Erstellen** eines ServiceRecords (SR)
2. **Konfigurieren** eines SR
3. **Registrieren** des SR in der Service Discovery Database

Bereitstellung eines Dienstes

1. **Erstellen** eines Service Records
   - Durch Aufruf der statischen Methode **Connector.open(String conStr)**
   - conStr: Uniform Resource Identifier (URI)
     - Beispiel: „btspp://localhost:4815162342…„
       - Protokoll
       - Hostadresse
       - UUID
   - Gibt **Connection-Objekt** zurück

Bereitstellung eines Dienstes

2. **Konfigurieren** eines Service Records
   - ServiceRecord wird implizit beim Erstellen durch die Methode Connector.open() mit richtigen Parametern gefüllt
   - Weitere Parameter können optional gesetzt oder verändert werden

Bereitstellung eines Dienstes

3. **Registrieren**
   - Durch Aufruf der Methode **acceptAndOpen()** des Connection-Objekts wird Service Record in der Service Discovery Database **registriert**
   - Ab jetzt kann eine **Verbindung zum Gerät** (Server) hergestellt werden

Programmcode: Server

```
try{
     //1.Erstellen eines ServiceRecords
     //2.Konfigurieren des SR
     StreamConnectionNotifier scn = (StreamConnectionNotifier)Connector.open
        ("btspp://localhost:4815162342");//UUID gekürzt

     //3.Registrieren des SR in der SDDB
     StreamConnection con = scn.acceptAndOpen();

       //Wenn Verbindung zum Client steht, wird dieser Programmteil abgearbeitet
       communicate(con);//Datenaustausch
   } catch (IOException ioe)
   {
      //Verbindung kann nicht geöffnet werden
   } catch (ServiceRegistrationException sre)
   {
      //Dienst kann nicht registriert werden
   }
```

3.Gerätesuche

Arten von Gerätesuche

- general inquiry
  - Alle Geräte in der Umgebung
- limited inquiry
  - Geräte die fest im BCC eingetragen sind
  - Geräte die bei vorheriger Suche gefunden wurden

Klasse: DiscoveryAgent

- Instanz holen durch Aufruf von **getDiscoveryAgent()** auf LocalDevice-Instanz

general inquiry – vollständige Suche

- DiscoveryAgent-Instanz
  - **startInquiry (int accessCode, DiscoveryListener dl)**
    - **accessCode**:
      Discoverable Mode der zu findenden Geräte
      - DiscoveryAgent.GIAC
      - DiscoveryAgent.LIAC
    - **DiscoveryListener:**
      für die Verarbeitung der Ergebnisse notwendig

general inquiry – vollständige Suche

- Interface **DiscoveryListener**
  - deviceDiscovered(RemoteDevice, DeviceClass)
    - Gerät wurde gefunden
  - inquiryCompleted(int type)
    - Suche ist abgeschlossen
    - Type - 3 Möglichkeiten (DiscoveryListener-Konstanten): abgeschlossen, abgebrochen, Fehler aufgetreten
- Dem Programmierer bleibt überlassen, wie auf diese Events reagiert werden soll

Programmcode: general inquiry

```
//DiscoveryAgent holen
DiscoveryAgent agent = dev.getDiscoveryAgent();//dev = LocalDevice
//Gerätesuche über agent starten
agent.startInquiry(DiscoveryAgent.GIAC, new DiscoveryListener(){

    public void deviceDiscovered(RemoteDevice dev, DeviceClass cod) {
    //Gerät speichern (z.B.) in Vector devList
    devList.add(dev);
    }

    public void inquiryCompleted(int type) {
         //Wenn Suche erfolgreich abgeschlossen
         if (type == DiscoveryListener.INQUIRY_COMPLETED)
         {
           //Geräte einzeln nach Diensten durchsuchen
         }
    }

});
```

limited inquiry – beschränkte Suche

- DiscoveryAgent-Instanz
  - **retrieveDevices(int option)**
    - **option:**
      - DiscoveryAgent.CACHED
        - » Ergebnisse aus vorherige Suche
      - DiscoveryAgent.PREKNOWN
        - » Im BCC gespeicherte Geräte
    - Gibt Array aus **RemoteDevices** zurück

Programmcode: limited inquiry

```
//DiscoveryAgent holen
DiscoveryAgent agent = dev.getDiscoveryAgent();//dev = LocalDevice

//Geräte einer vorherigen Suche abrufen
RemoteDevice[] rdc = agent.retrieveDevices(DiscoveryAgent.CACHED);

//im BCC gespeicherte Geräte abrufen
RemoteDevice[] rdp = agent.retrieveDevices(DiscoveryAgent.PREKNOWN);

//Weitere Anweisungen, z.B. Geräte nach Diensten dursuchen
```

4.Dienstsuche

4.Dienstsuche

- Notwendig für den **Client**
- Ähnlich wie Gerätesuche
- Ebenfalls über DiscoveryAgent und DiscoveryListener
- Methode: **DiscoveryAgent.searchServices()**

DiscoveryAgent.searchServices()

- **searchServices(int[ ] attrSet, UUID[ ] uuidSet, RemoteDevice btDev, DiscoveryListener discListener)**
  - attrSet:
    ServiceRecord-Attribute, die DiscoveryAgent bei jedem gefundenen Service abfragen soll
  - uuidSet:
    UUIDs der gesuchten Dienste
  - btDev:
    zu durchsuchendes Gerät
  - discListener:
    DiscoveryListener, der Events der Suche verarbeitet

DiscoveryListener

- Zu implementierende Methoden:
  - servicesDiscovered(int transID, ServiceRecord[] servRecord)
    - servRecord:
      ServiceRecords der gefundenen Dienste
  - serviceSearchCompleted(int transID, int respCode)
    - respCode:
      Zeigt an, ob Suche erfolgreich war oder mit einem Fehler abgebrochen wurde
      - Vergleich mit Konstanten in DiscoveryListener

Programmcode: Client sucht

```
//DiscoveryAgent holen
DiscoveryAgent agent = dev.getDiscoveryAgent();//dev = LocalDevice
//UUID des zu findenden Dienstes speichern
UUID[] uuids = { new UUID("23432345675323456786544815162342", false) };
//Dienstsuche über agent starten
agent.searchServices(null, uuids, remoteDev, new DiscoveryListener(){//remoteDev
    aus Gerätesuche vorhanden


    //Services gefunden
    public void servicesDiscovered(int tID, ServiceRecord[] srvRec) {
          for(ServiceRecord rec : srvRec)//jeden ServiceRecord
             srVec.add(rec);//im Vector abspeichern
    }

    //Suche beendet
    public void serviceSearchCompleted(int tID, int respCode) {
          //Falls Suche normal beendet
          if (respCode == DiscoveryListener.INQUIRY_COMPLETED)
          {
             //Verbindungsaufbau
          }
    }
});
```

Verbinden zum Dienst

- Verbindung in **serviceSearchCompleted()** herstellen
  - Wenn respCode positiv war
- Nutzen des durch servicesDiscovered() abgespeicherten ServiceRecords:
  - String conStr = record.**getConnectionURL()**
- Connection String für die Verbindung verwenden:
  - **Connector.open(conStr)**

Programmcode: Client verbindet

```
public void serviceSearchCompleted(int tID, int respCode)
{
    //Falls Suche normal beendet
    if (respCode == DiscoveryListener.INQUIRY_COMPLETED)
    {
        //Verbindungsaufbau

        //Connection URL vom gefundenen Dienst holen
        String conStr = srVec.firstElement().getConnectionURL();
        //Verbindung aufbauen
        StreamConnection con = (StreamConnection)Connector.open(conStr);

        //Wenn verbunden, dann wird dieser Programmteil abgearbeitet
        communicate(con);
    }
}
```

5.Datenaustausch

Unterstützte Protokolle

- L2CAP:
  Logical Link Control and Adaption Protocol
- OBEX:
  Object Exchange-Protocol
- **SPP:**
  Serial Port Profile (RFCOMM)
- Weitere Protokolle können unter Zuhilfenahme der Vorhandenen implementiert werden

Serial Port Profile

- Emuliert eine serielle Schnittstelle mit Hilfe des RFCOMM-Protokolls
- **Einfache Datenströme**
  - **StreamConnection**-Klasse (für SPP)
- Protokollteil des Connection-Strings: „btspp://“ – *BlueTooth Serial Port Profile*

Datenaustausch

- Grundlage:
  - **<u>Connection-Objekt</u>**
    (StreamConnection bei SPP)
- Server:
  Nach dem Registrieren eines Dienstes
- Client:
  Nach erfolgreicher Suche des Dienstes

StreamConnection

- Wichtige **Methoden**:
  - openInputStream()
  - openOutputStream()
- Kommunikation über **read()** und **write()** der Streams
- Server und Client umgekehrt über Streams zugeordnet

Programmcode: Server

```
void communicate (StreamConnection con) {
    try {
        //Streams holen
        InputStream in = con.openInputStream();
        OutputStream out = con.openOutputStream();

        //Hallo senden
        out.write(new String("Hallo").getBytes());

        //Verbindung beenden
        in.close(); out.close(); con.close();

    } catch (IOException ioe) {
      //Daten konnten nicht gesendet werden...
    }
}
```

Programmcode: Client

```
void communicate (StreamConnection con) {
      try {
      //Streams holen
      InputStream in = con.openInputStream();
      OutputStream out = con.openOutputStream();

      //Text empfangen
      int ch;
      StringBuilder text = new StringBuilder();
      while ( (ch = in.read()) != -1) {//-1 = Ende des Strings
            //Char empfangen und speichern
            text.append((char)ch);
      }

      //String auf dem Display ausgeben

      //Verbindung beenden
      in.close(); out.close(); con.close();

      } catch (IOException ioe) {
         //Daten konnten nicht empfangen werden...
      }
}
```

Quellen

- http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/
- http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth2/
- http://www.wi.uni-muenster.de/pi/lehre/ws0607/skiseminar/ausarbeitungen/06-Bluetooth.pdf
- JavaDoc der Bluetooth APIs

- Bluetooth APIs sind enthalten im Wireless Toolkit 2.5 for CLDC:
- http://java.sun.com/products/sjwtoolkit/download-2_5.html

ENDE