Einführung in CloudKit

Friedrich Ruynat

Use-Cases

Cocoaheads-App

- Termine ankündigen (+ Reminder)
- Vortragsfolien verfügbar machen
- Andere Cocoaheads suchen & finden
- Push-Notifications für Terminänderungen
- App für iOS und macOS
- Website für Externe

Notiz-App

- Notizen mit Bild-Anhängen
- Offline und Online-Editieren
- Sync zwischen Mac, iPad, iOS, Web
- Sharing von Notizen
  - Notification für Änderungen
  - Konfliktbehandlung

Server, PaaS, …

- ⊖ Eigenes Backend entwickeln
- ⊖ Eigener Login notwendig
- ⊖ Sicherheit und Privacy
- ⊖ Verfügbarkeit und Wartung
- ⊖ Push Notifications…
- ⊖ Betriebskosten

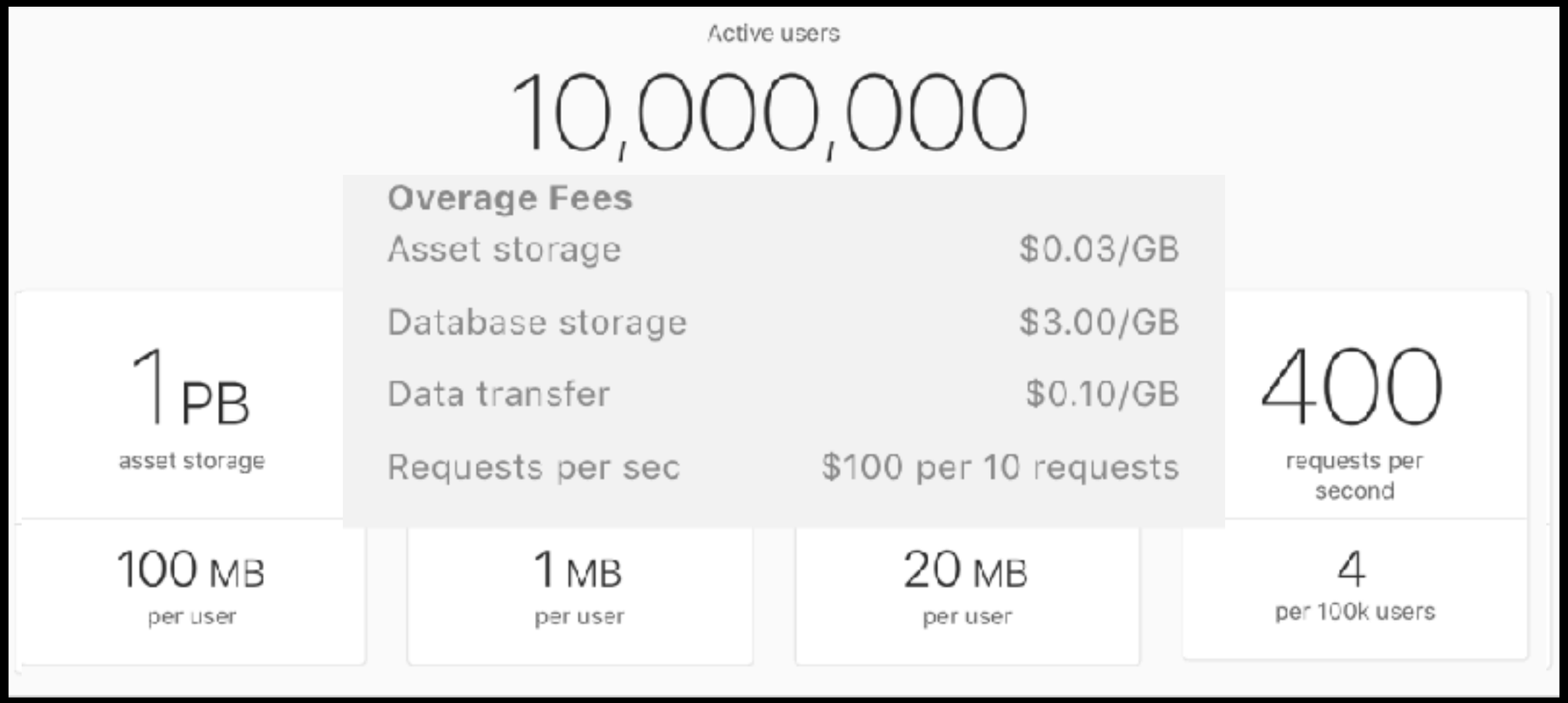
Alternative CloudKit

- Speicherung von Daten auf iCloud-Servern
- Öffentliche und private Datenbanken + Sharing
- Nutzt iCloud-Accounts (iOS: über 90% iCloud-Nutzer)
- Öffentliche Daten: Ohne Account nutzbar!
- User-Discovery über iCloud (Datenschutzfreundlich…)
- Live-Updates über Push Notifications
- Web / Android: CloudKit JS, Server-to-Server-API

Kostenlos?

Apple: Missbrauch verhindern.

- **Private / Geteilte Daten:** Quota vom iCloud-Account des Nutzers
- **Öffentliche Daten:** Je nach Nutzerzahl



Struktur

Container (pro App)

Private Database

Record Zones

Record

Default Zone

Record

Private Daten des Nutzers

Public Database

Default Zone

Record

Erfordert kein iCloud-Account

Öffentliche Daten (z.B. News, Templates, …)

Shared Database

Shared Zones

Record

Geteilt zwischen einzelnen Nutzern

CKRecord

- Key/Value-Paare
- Referenzierbarkeit: Record Identifier (Custom / UUID)
- Change-Token (später)
- Record Type:
  - Schema-Definition (Key → Value Type)
  - Zugriffsrechte

Record Value Types

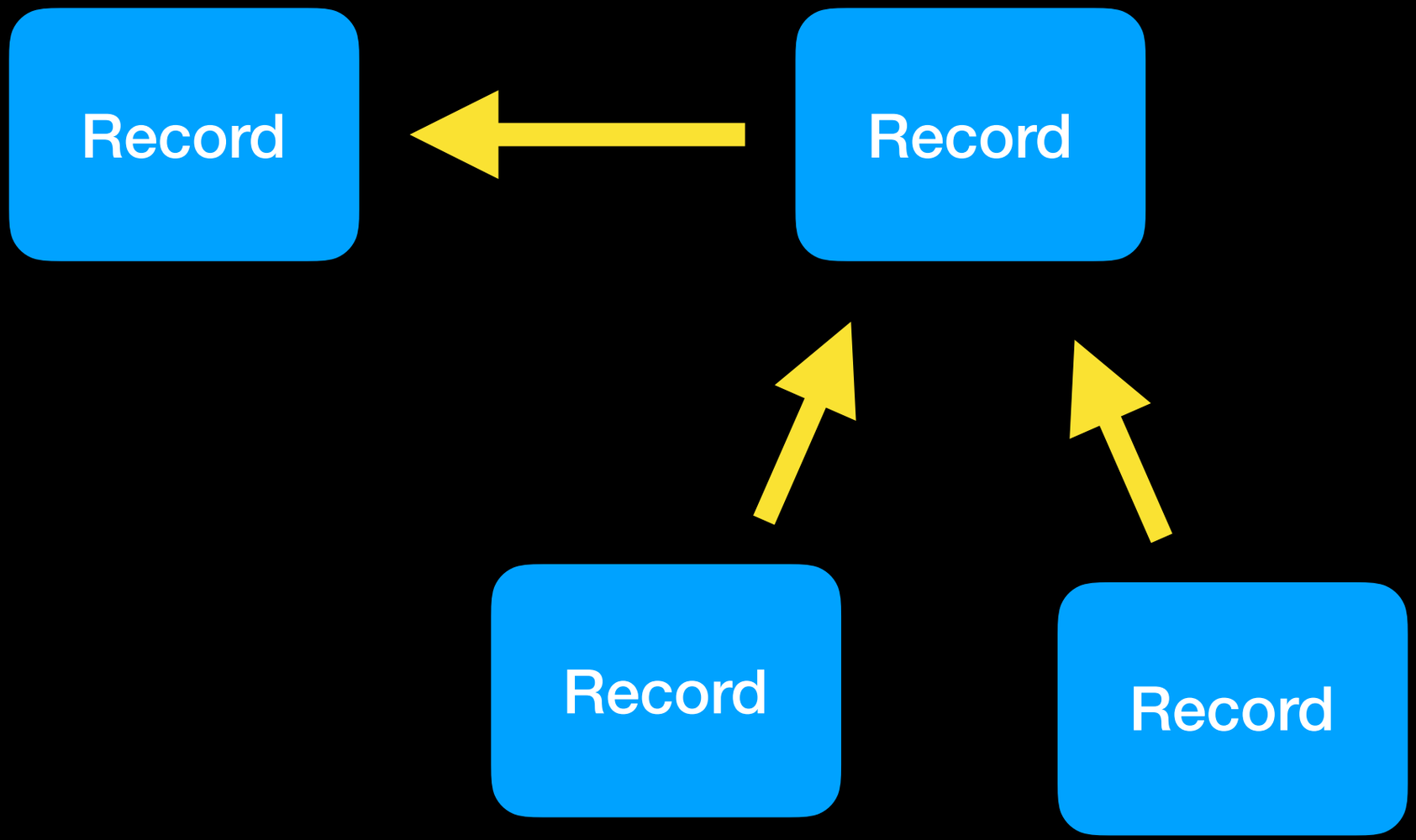
- NSString, NSNumber, NSDate, NSData
- Arrays
- **CLLocation**: Umkreissuche etc.
- **CKAsset**
- **CKReference**

CKAsset

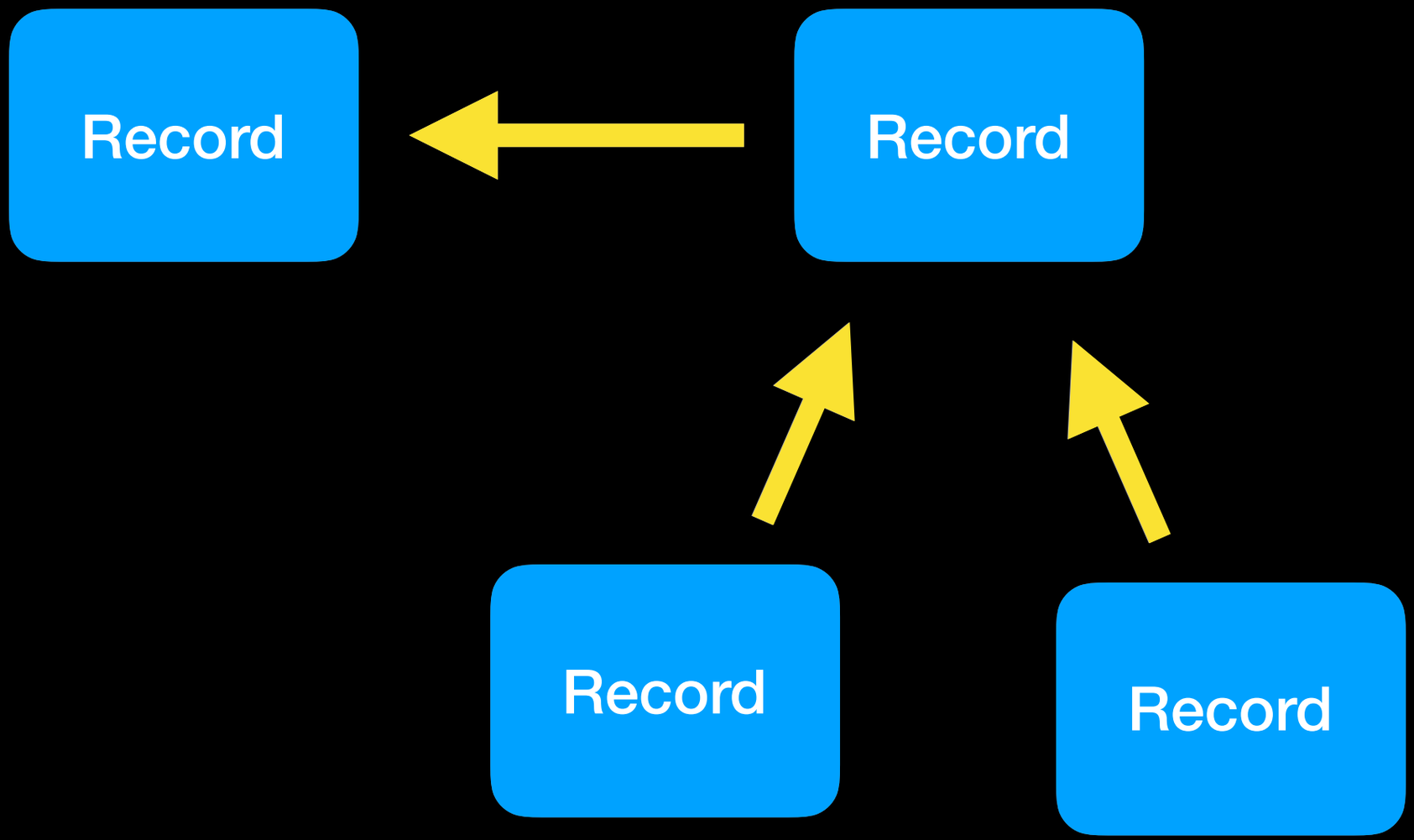
- BLOBs (Bilder, Audiodaten etc.)

- Effiziente Uploads / Downloads
- Gemeinsame Nutzung zwischen Records
- Garbage Collection

CKReference



Kaskadierendes Löschen (Optional)



Environments

- **Developer-Environment:** Requests erzeugen Schema
- **Production-Environment:** Festes Schema
- **CloudKit Dashboard**
  - Deployment Development → Production
  - Bearbeiten Datensätzen
  - Public Database: Rollen-Verwaltung

Zugriffsrechte

- **Public Database**: Lesen + Schreiben mit/ohne Account
- **Record Type:** Zugriffsrechte (Create, Read, Write)
- Zugriffsrechte nach Rollen gruppiert
- **Default Roles:** World, Authenticated, Creator
- **Custom Roles**: CloudKit Dashboard
  - Frei definierbar
  - Zuweisung zu konkreten Nutzern nur im Dashboard

Caching

- Wie werden Daten lokal gespeichert?
- CloudKit: 🤷‍♀️

- **Reine** Transport-API mit Fetch und Store
- **Kein** Sync von Server und Client
- App für lokales Caching verantwortlich
- Serialisierung: NSCoder
- Updates: Queries, Subscriptions (später)

Operationen

Operationen

- Convenience API
- CKOperation API
  - NSOperation: Quality-of-Service etc.
  - Langlebige Operationen (je QoS bis zu 7 Tage)

Convenience: Save

```
let db = CKContainer.defaultContainer.privateCloudDatabase

let drinkID = CKRecordID(recordName: "CubaLibre")
let drink = CKRecord(recordType: "Drink", recordID: noteID)

drink["name"] = "Cuba Libre"
drink["asset"] = CKAsset(…)

db.saveRecord(drink) { savedRecord, error in
    …
}
```

Convenience: Fetch

```
let db = CKContainer.defaultContainer.privateCloudDatabase

let drinkID = CKRecordID(recordName: "CubaLibre")


db.fetchRecord(drinkID) { fetchedRecord, error in
    …
}
```

Queries

Erfordern Index

```
let db = CKContainer.defaultContainer.privateCloudDatabase

let predicate = NSPredicate(format: "name BEGINSWITH 'Cuba'")
let query = CKQuery(recordType: "Drink", predicate: predicate)

db.performQuery(query, inZoneWithID: nil) { results, error in
    …
}
```

Fehlerbehandlung

- Behandeln von Fehlern ist **kritisch**
- **Dauerhafte Fehler:** incompatibleVersion
- **Temporäre Fehler:**
  networkUnavailable, requestRateLimited, zoneBusy, …
- **Nutzer-seitigte Fehler:**
  NotAuthenticated, permissionFailure, quotaExceeded, …
- **Teilweise Fehler** (Batch-Operationen, Mehrere Items)

Konflikte

Alice iPad

Lokale Kopie (Change: 0x1)	
Titel	Eröffnung BER
Datum	3.6.2012

Offline-Änderung

Lokale Kopie (Change 0x1)	
Titel	Eröffnung BER
Datum	27.10.2013

Fetch

iCloud-Sever

Record (Change: 0x1)	
Titel	Eröffnung BER
Datum	3.6.2012

Fetch

Bob Mac

Store

Record (Change: 0x2)	
Titel	Eröffnung BER
Datum	17.3.2013

Store

0x1 != 0x2

CKErrorServerRecordChanged

Konflikt: serverRecordChanged

- Client: Lädt Änderungen an veralteter Version hoch
- Server: Client soll Konflikt beheben
- NSError.userInfo: Anchestor-, Server-, Client-Record
- Client merged, aktualisiert ServerRecord
- Triviale Fälle: Alternative Policies (z.B. alles überschreiben)

Atomare Operationen

- Änderungen mehrerer Records
- Objekt-Graph mit Referenzen
- CKModifyRecordsOperation.isAtomic
- Nur innerhalb einer Record Zone
- Nicht in der Default Zone

Subscriptions

Subscriptions

- **Polling**: Ineffizient + Rate Limiting
- **Subscription:** Push-Notifications für Änderungen
- **Sichtbare Notifications:** Titel, Badge, Sound usw.
    - Erfordert Nutzererlaubnis
    - Bei terminierter App
- **Silent Notifications:** Im Hintergrund
    - App darf nicht beendet werden

CKSubscription

- **CKDatabaseSubscription**: Änderungen einer Datenbank
- **CKRecordZoneSubscription:** Änderung einer Zone
- **CKQuerySubscription**: Änderungen über Query
- Handling wie Push-Notification über App Delegate
- Mehrere Änderungen in einer Notification

User-Records

User Accounts

- Public Database: CKRecordTypeUserRecord
- Public Readable: Erfordert aber Zustimmung
- Identity:
  - Privacy: Random-Identifier statt E-Mail-Adresse
  - Identifier stabil
  - Anderer Identifier pro Container

User Discovery

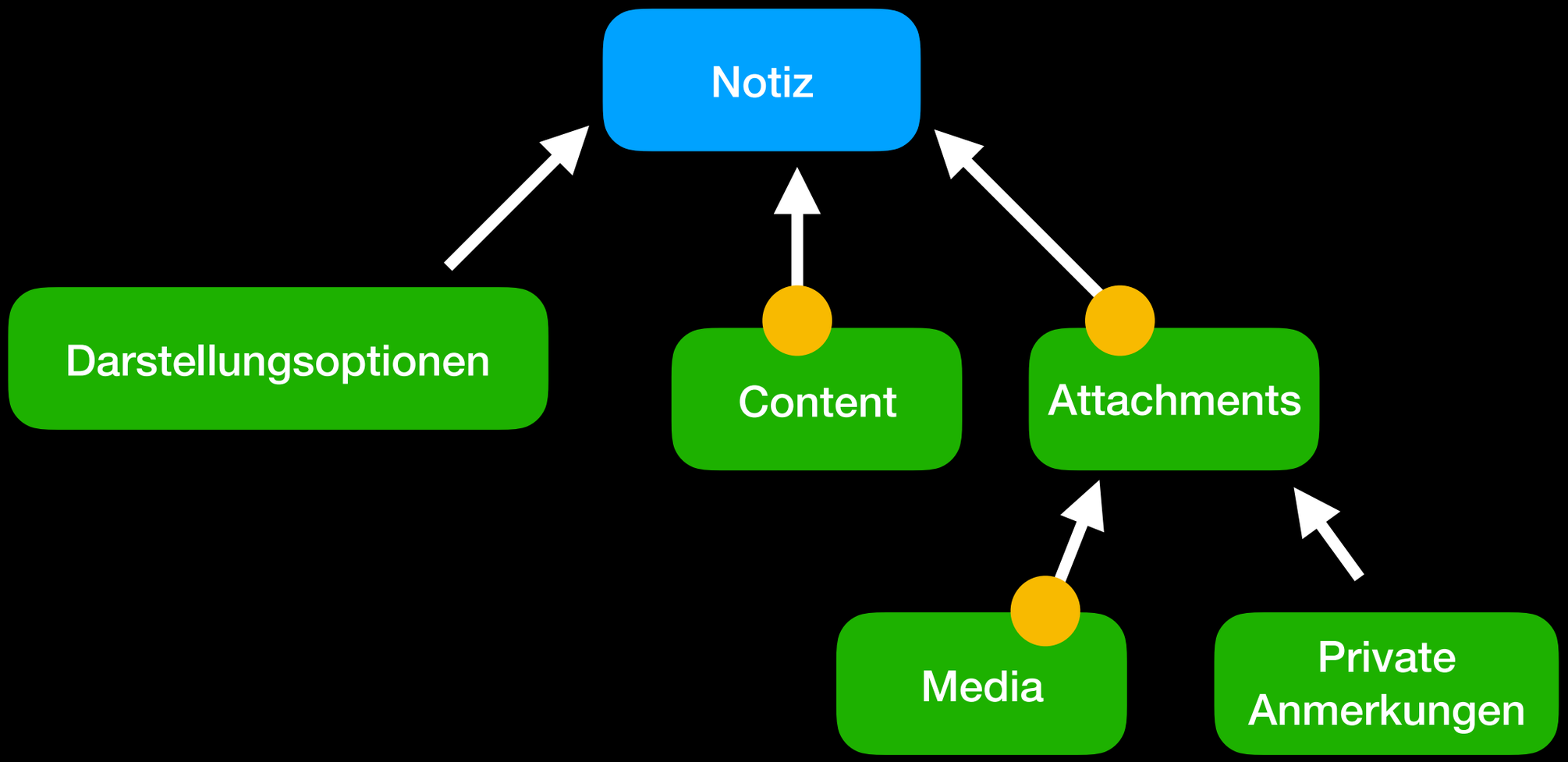
- Suche nach befreundeten Nutzern:
  - Abfrage über spezifische E-Mailadresse
  - Suche nach Nutzern im Adressbuch
- Nutzer müssen pro App zustimmen
- App benötigt keinen Zugriff auf Adressbuch

Sharing

Sharing

- CloudKit erlaubt Freigabe von Records
- Owner versendet Einladung
    - Zugriffsrechte Wählbar: Read / Write
    - Zugriff für bestimmten oder alle iCloud-Nutzer
    - Share-Sheet (iMessage, Mail, Pasteboard, …)
- Empfänger erhält URL
- URL öffnen: App erhält Share über App-Delegate

Sharing: Sichtbarkeit



CKRecord: Spezielle "Parent" Referenz

Sharing: Nur Parent-Referenzen werden geteilt

Zusammenfassung

- CloudKit: Transport / Storage iCloud-Server
- Kein eigenes Hosting notwendig
- Nutzung von iCloud-Accounts
- Fokus aus Privacy
- Sync: Nur unterstützende Funktionen