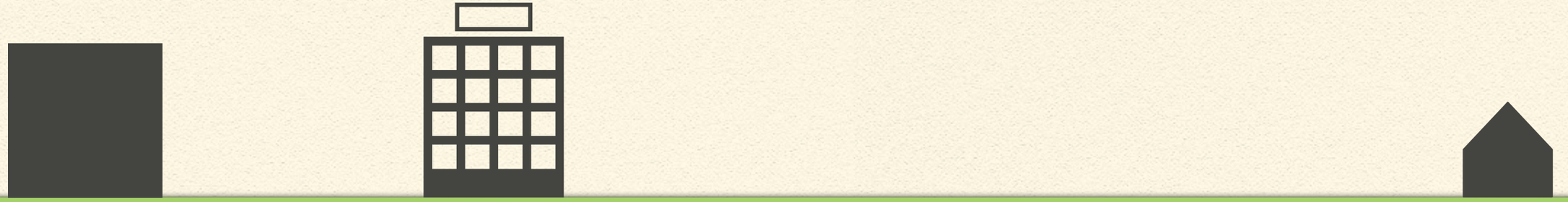
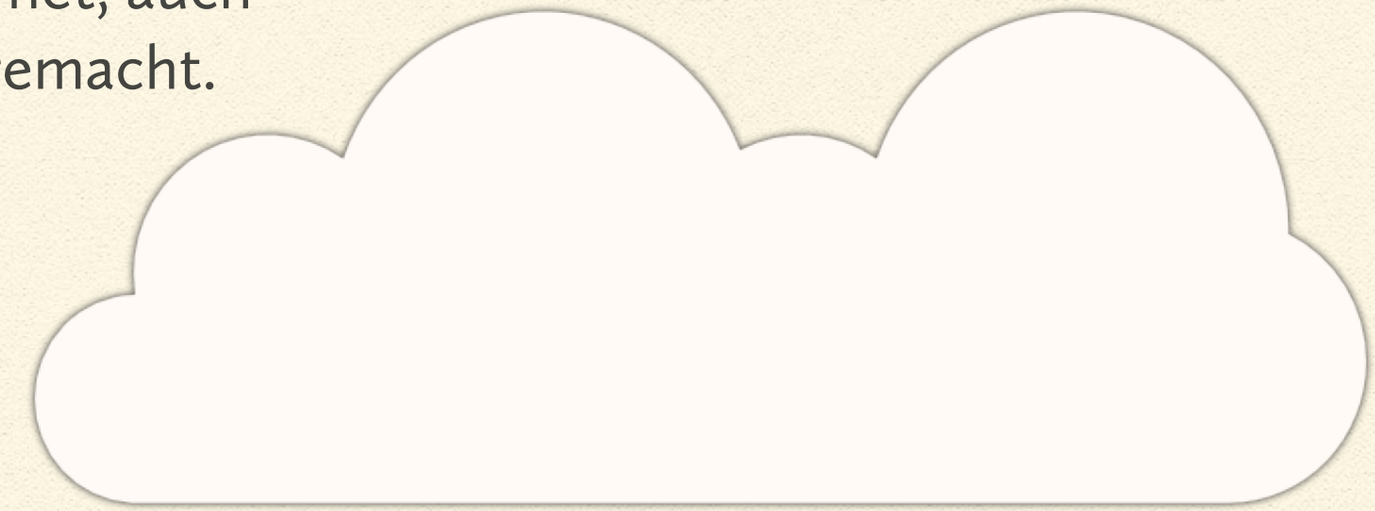
Flowy Apps erzählt eine kurze Geschichte über

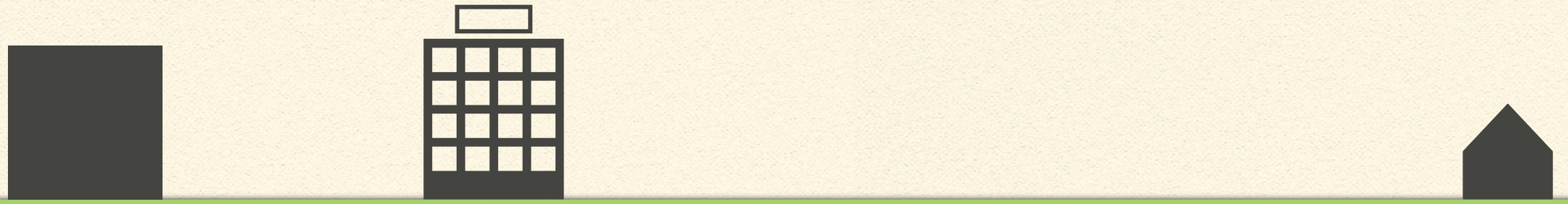
REDS

Remotely Encrypted Distributed Storage

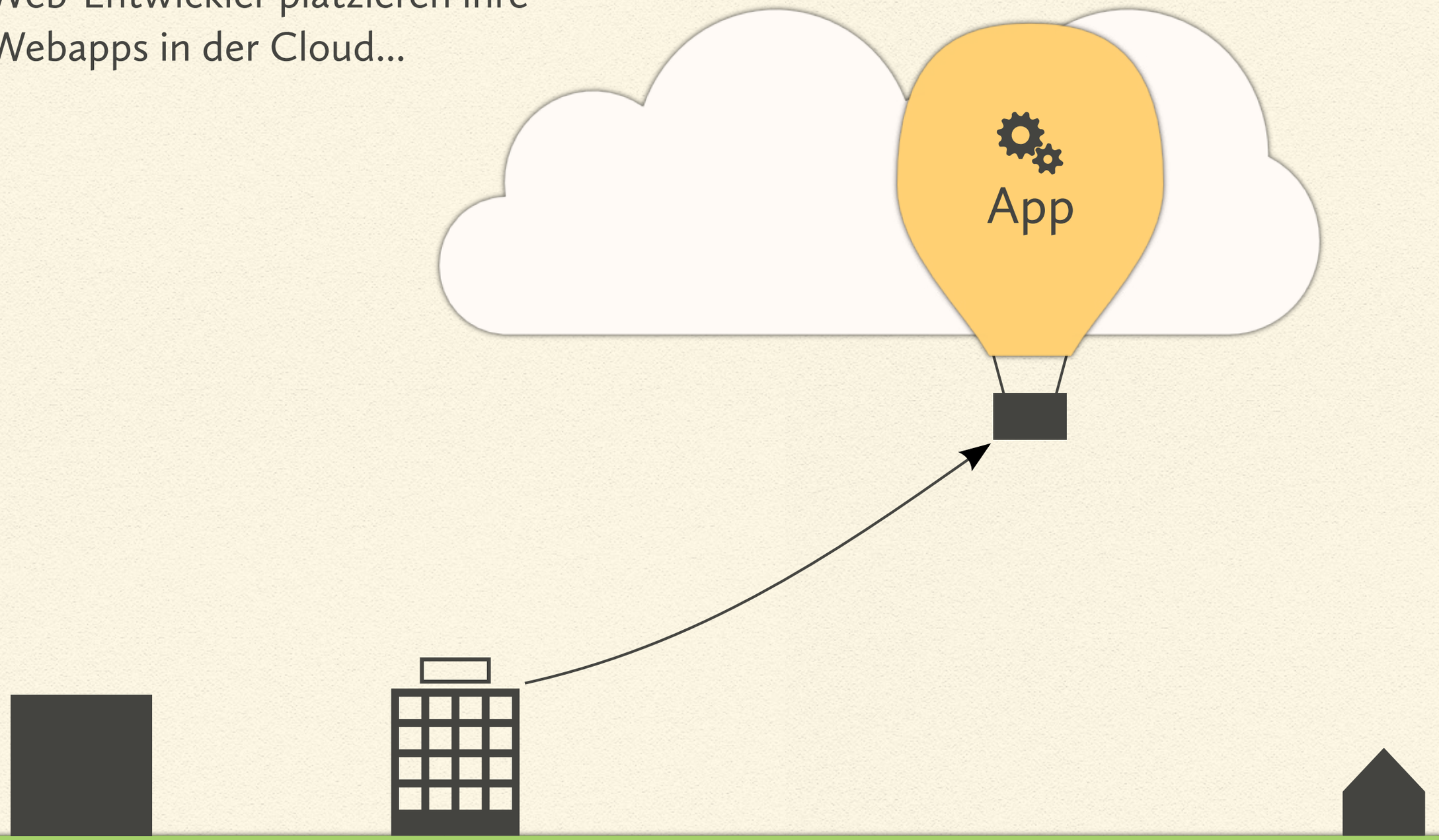


Heute wird alles im Internet, auch bekannt als die Cloud, gemacht.

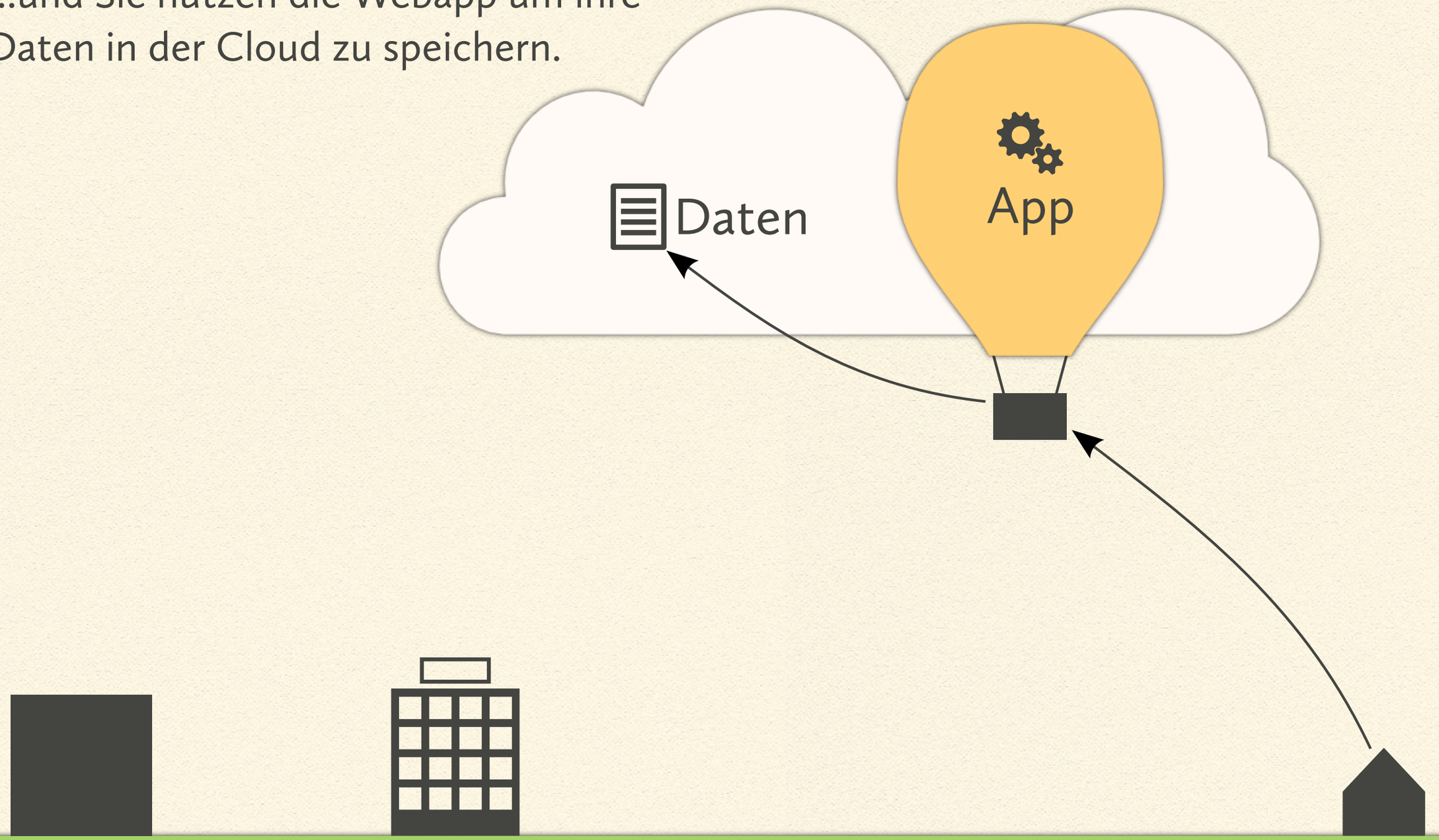




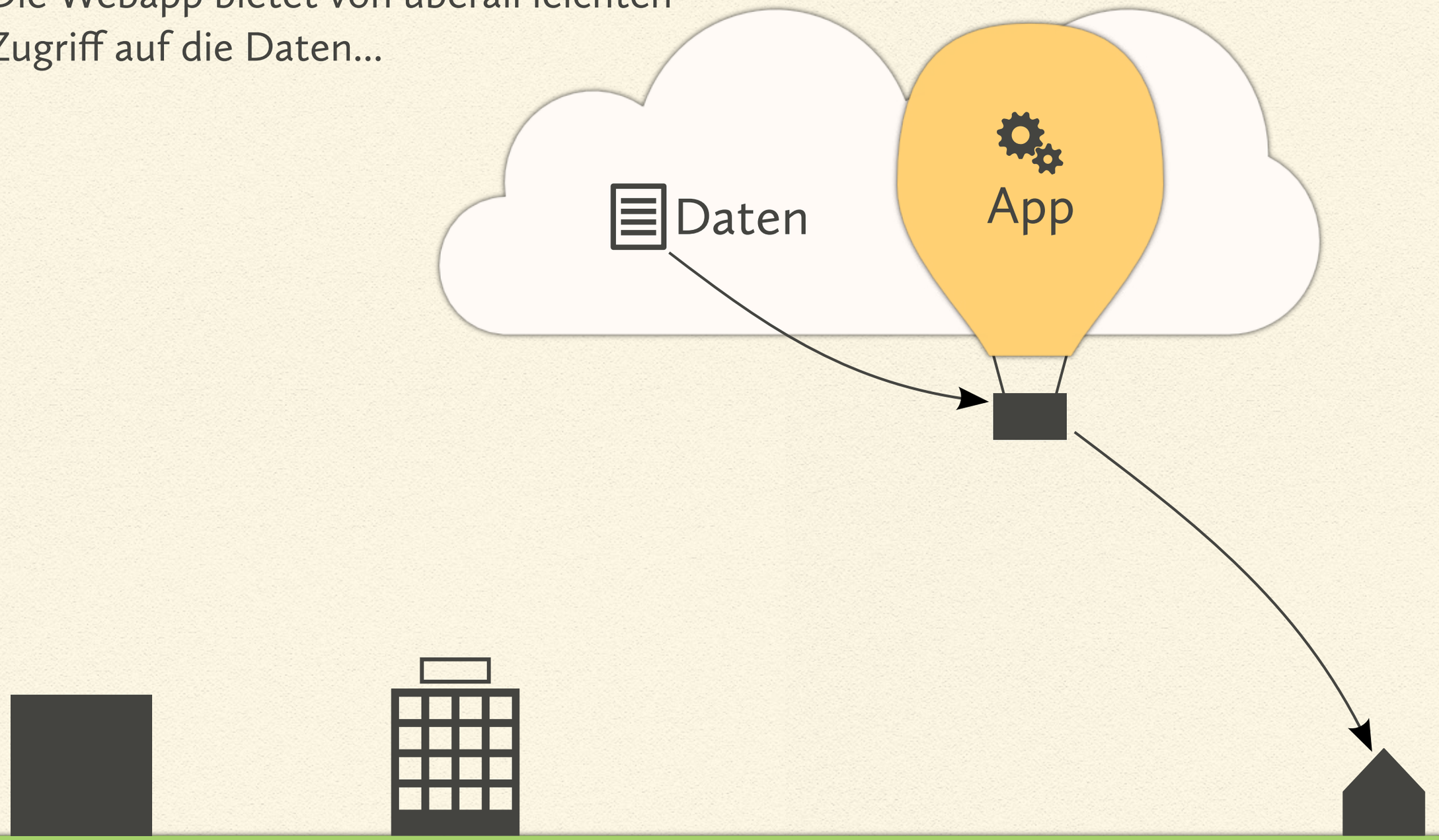
Web-Entwickler platzieren ihre Webapps in der Cloud...



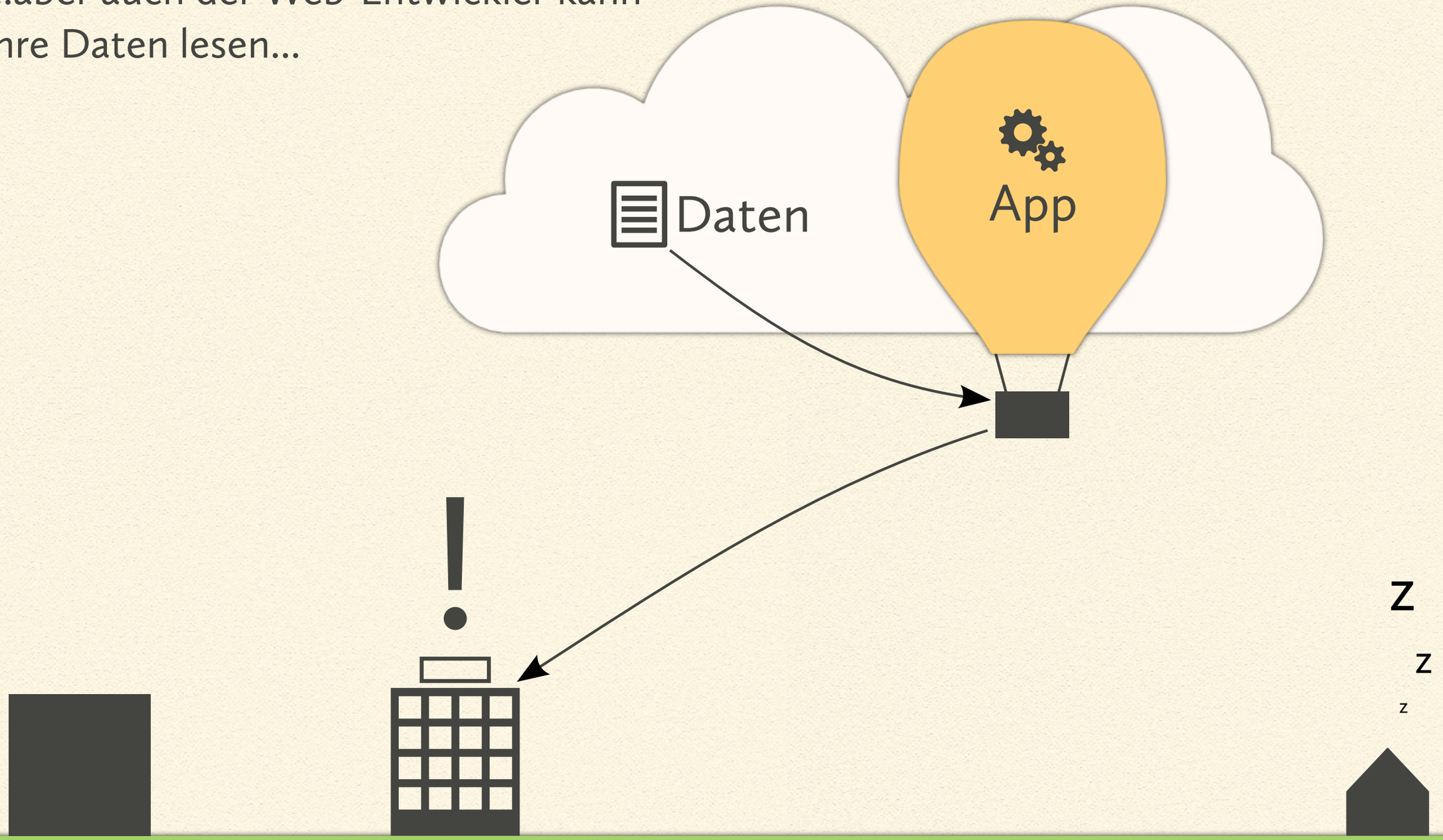
...und Sie nutzen die Webapp um ihre Daten in der Cloud zu speichern.



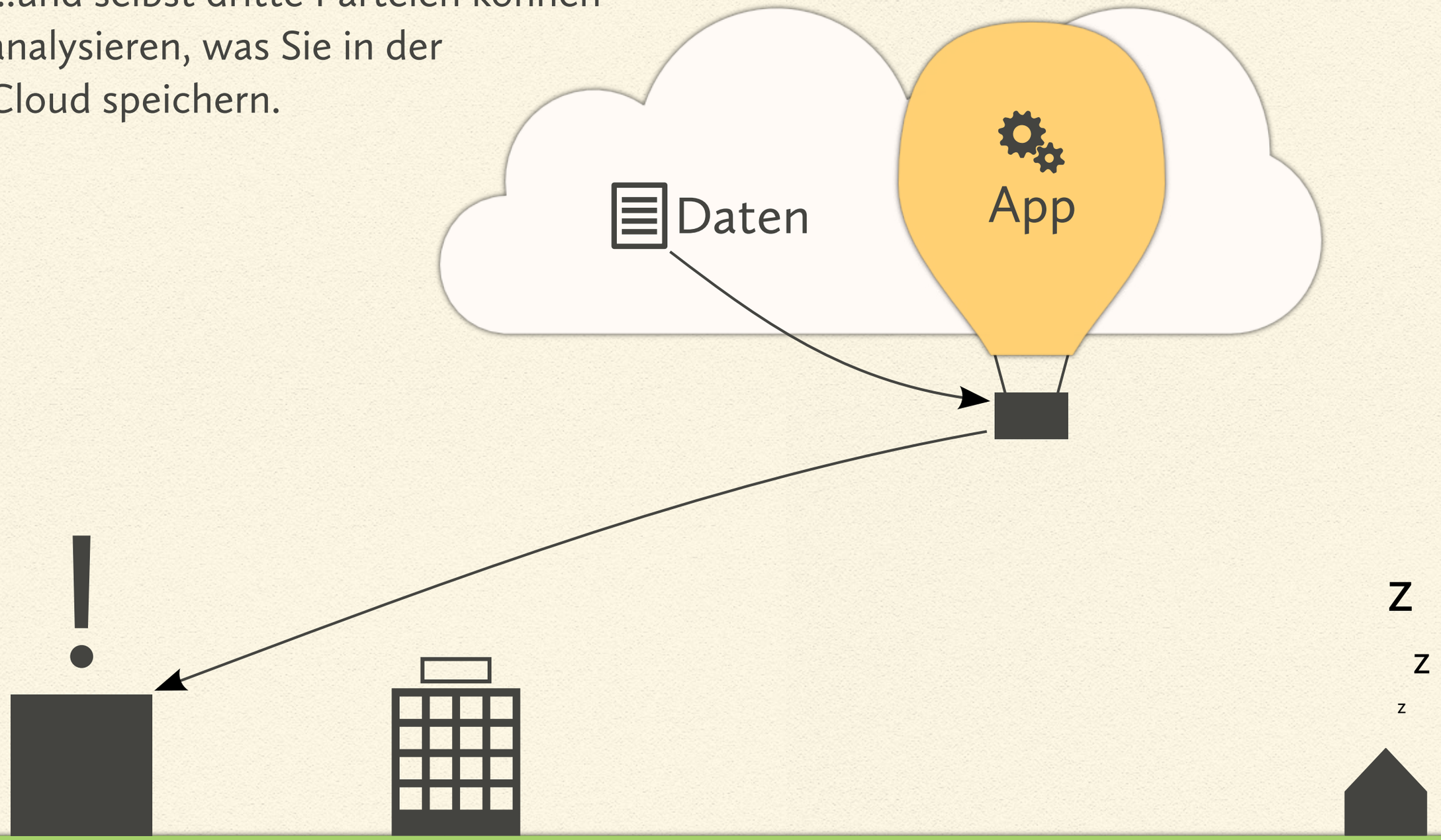
Die Webapp bietet von überall leichten Zugriff auf die Daten...



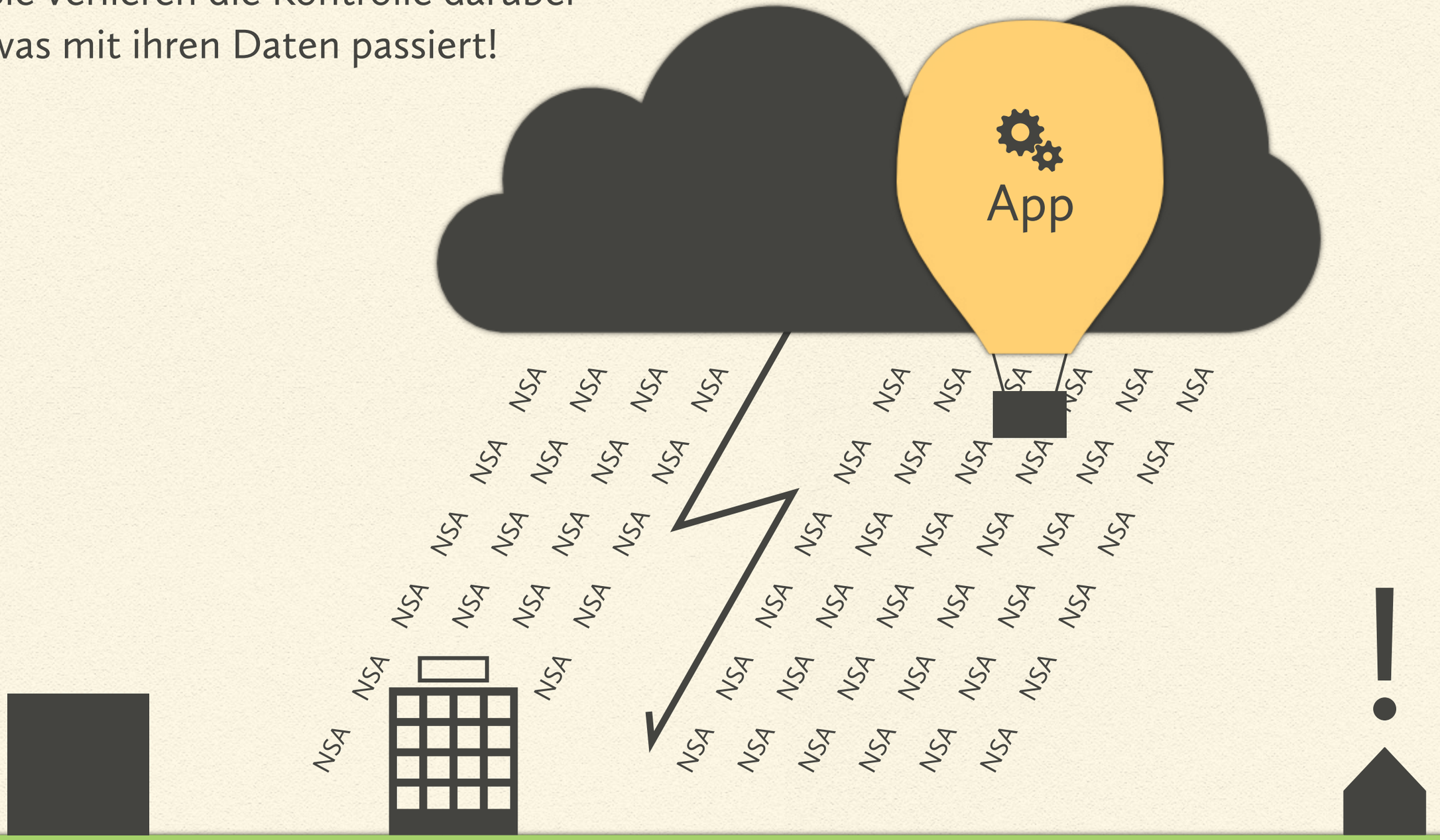
...aber auch der Web-Entwickler kann ihre Daten lesen...



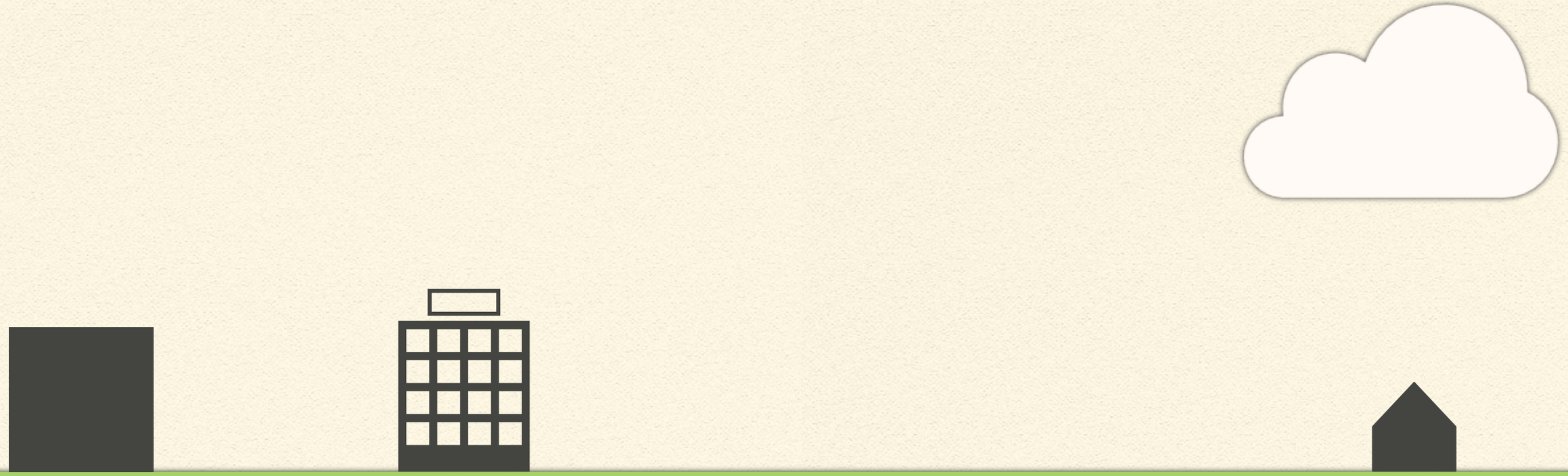
...und selbst dritte Parteien können analysieren, was Sie in der Cloud speichern.



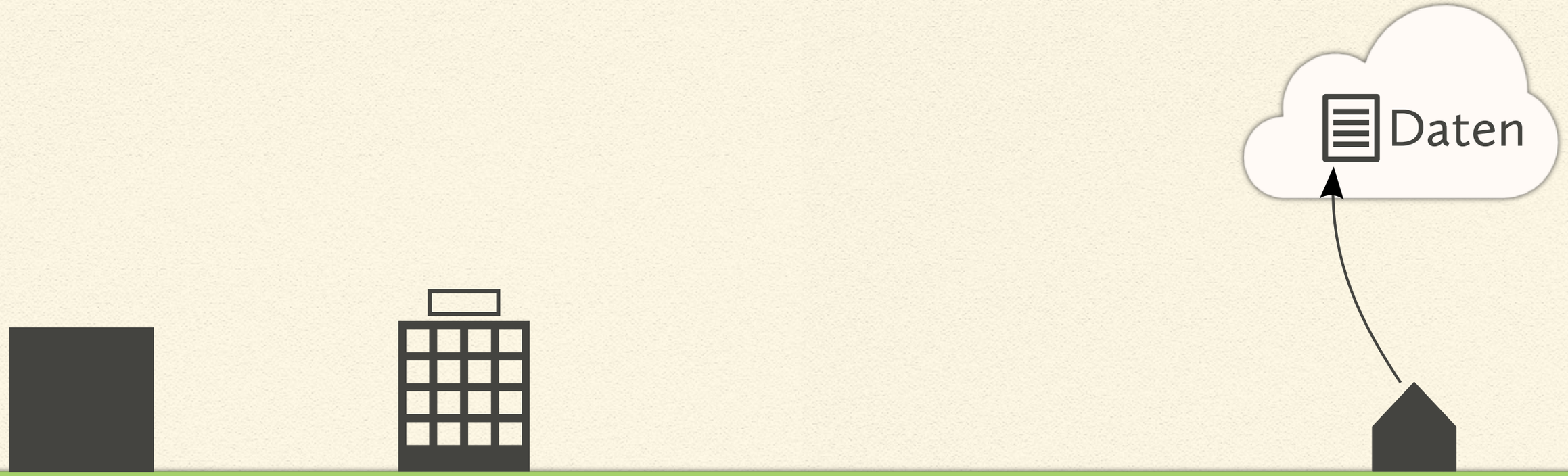
Sie verlieren die Kontrolle darüber
was mit ihren Daten passiert!



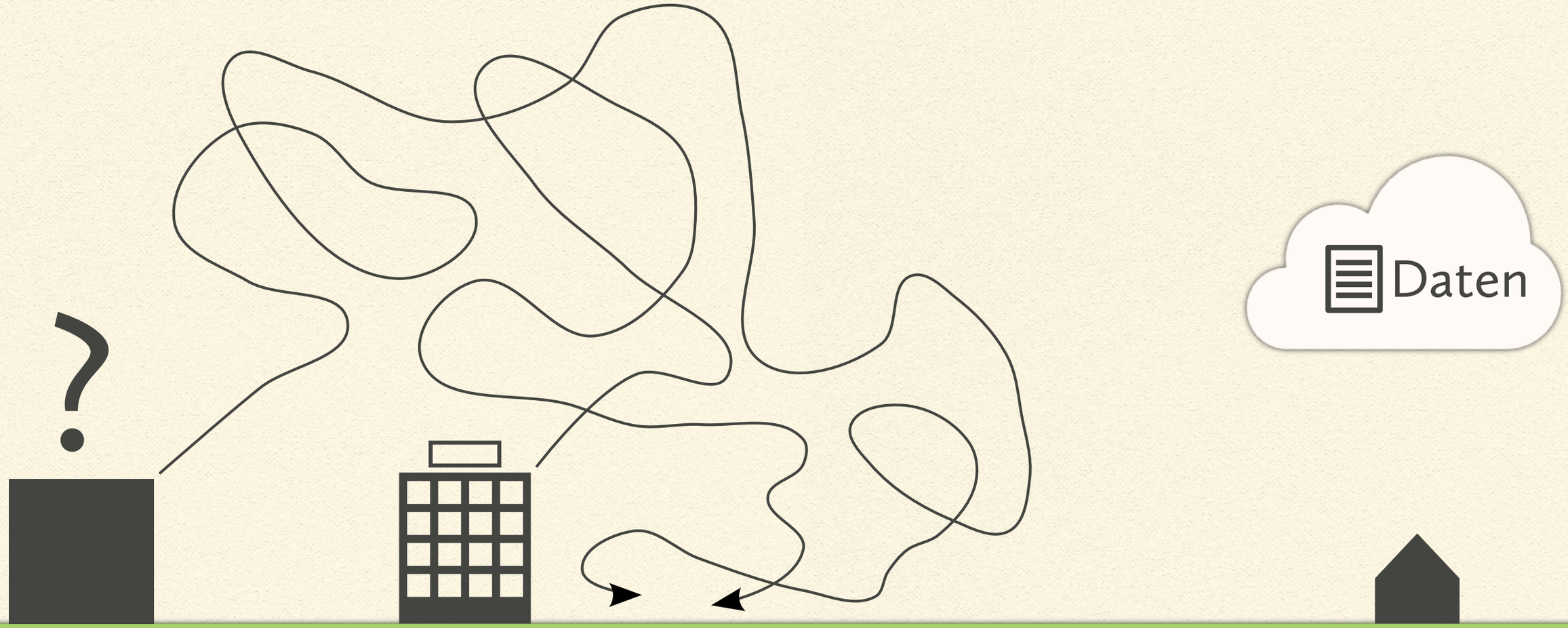
Eine eigene Cloud zu besitzen könnte das Problem lösen.



Sie speichern die Daten in ihrer eigenen Cloud...

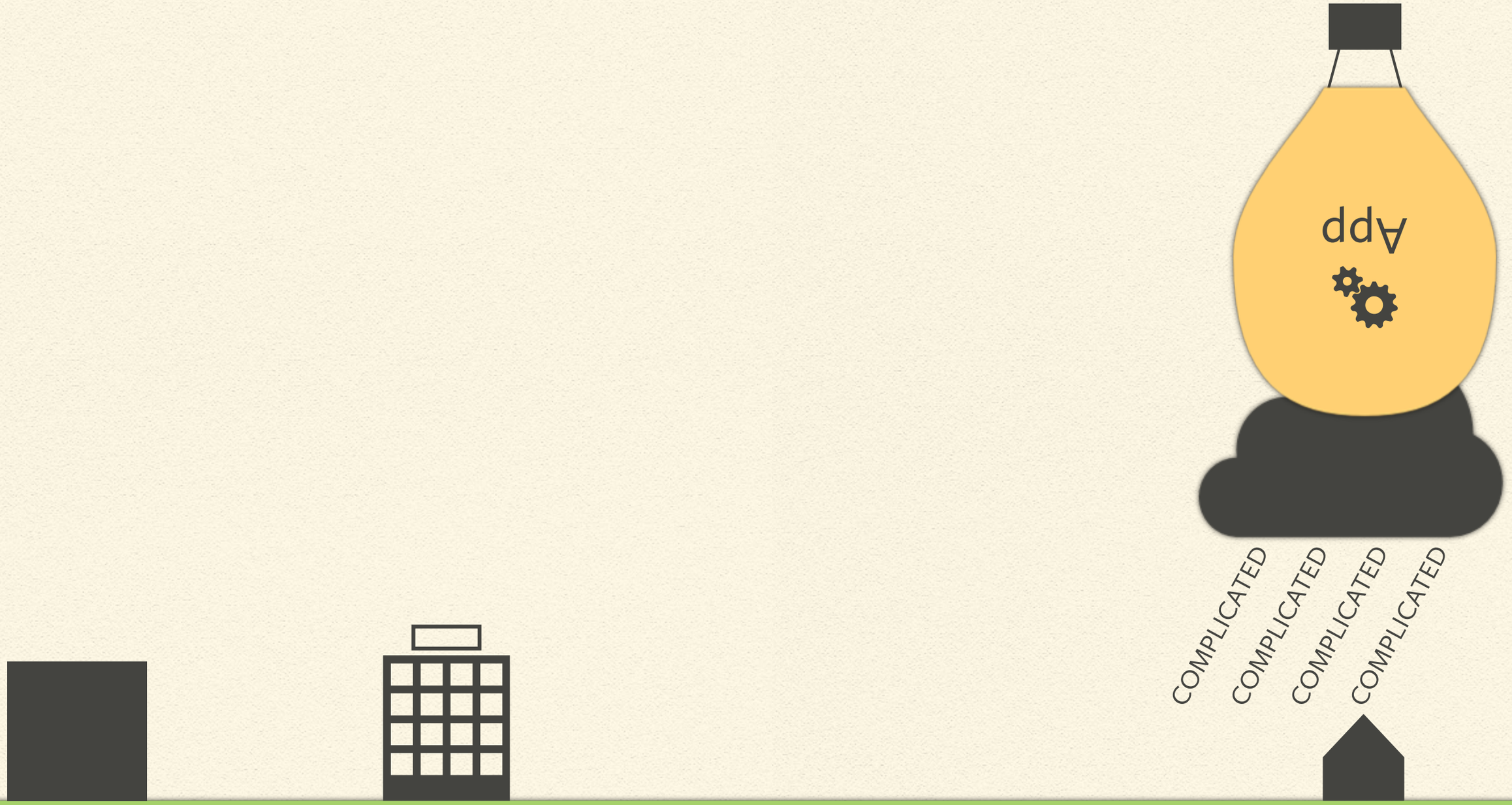


...und alle anderen haben keinen Zugriff auf ihre Daten.

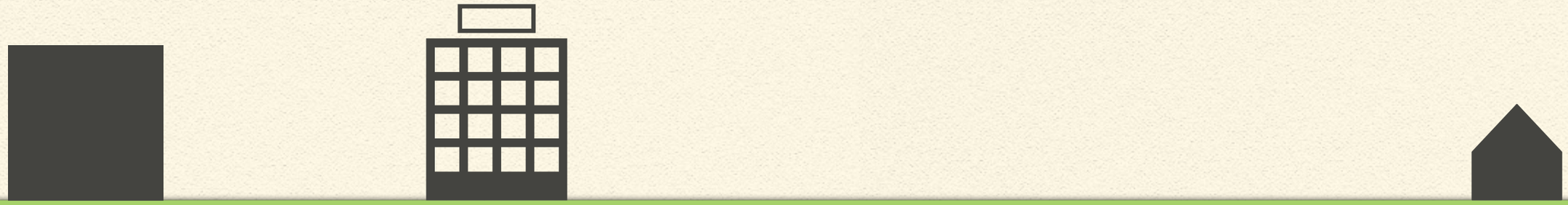


Aber Apps in der eigenen Cloud zu installieren
ist ziemlich kompliziert...

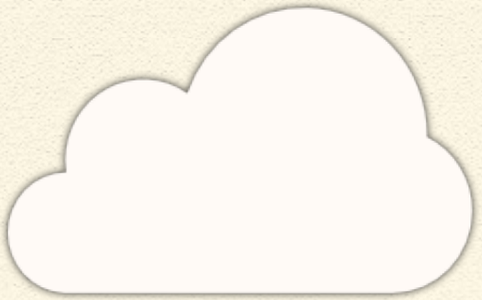
...also ist das auch nicht die beste Lösung.

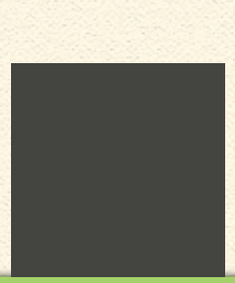


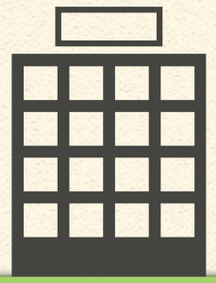
Also überdenken wir das Ganze mal...



Was wäre, wenn sie ihre eigene Cloud behalten könnten...

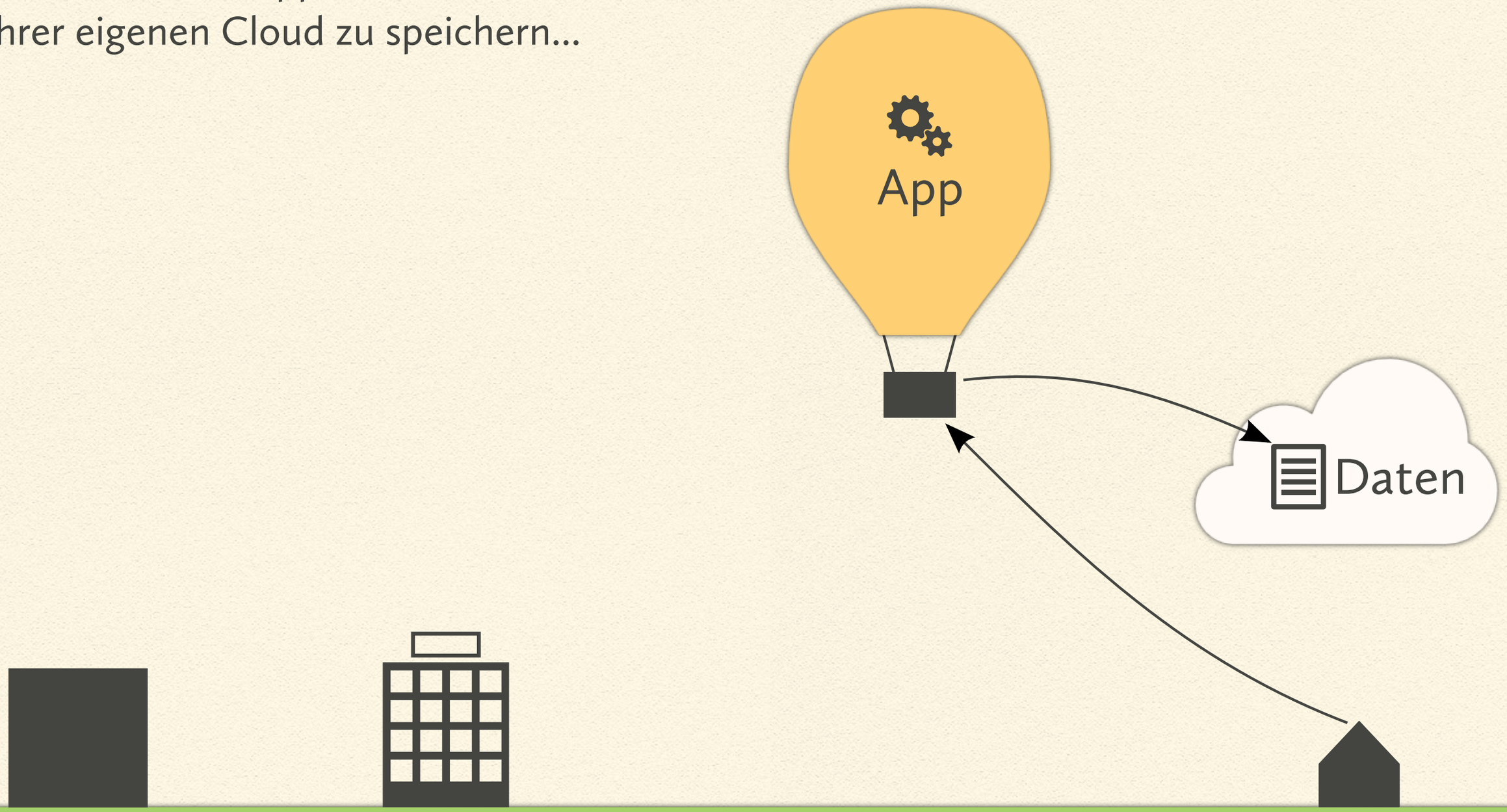




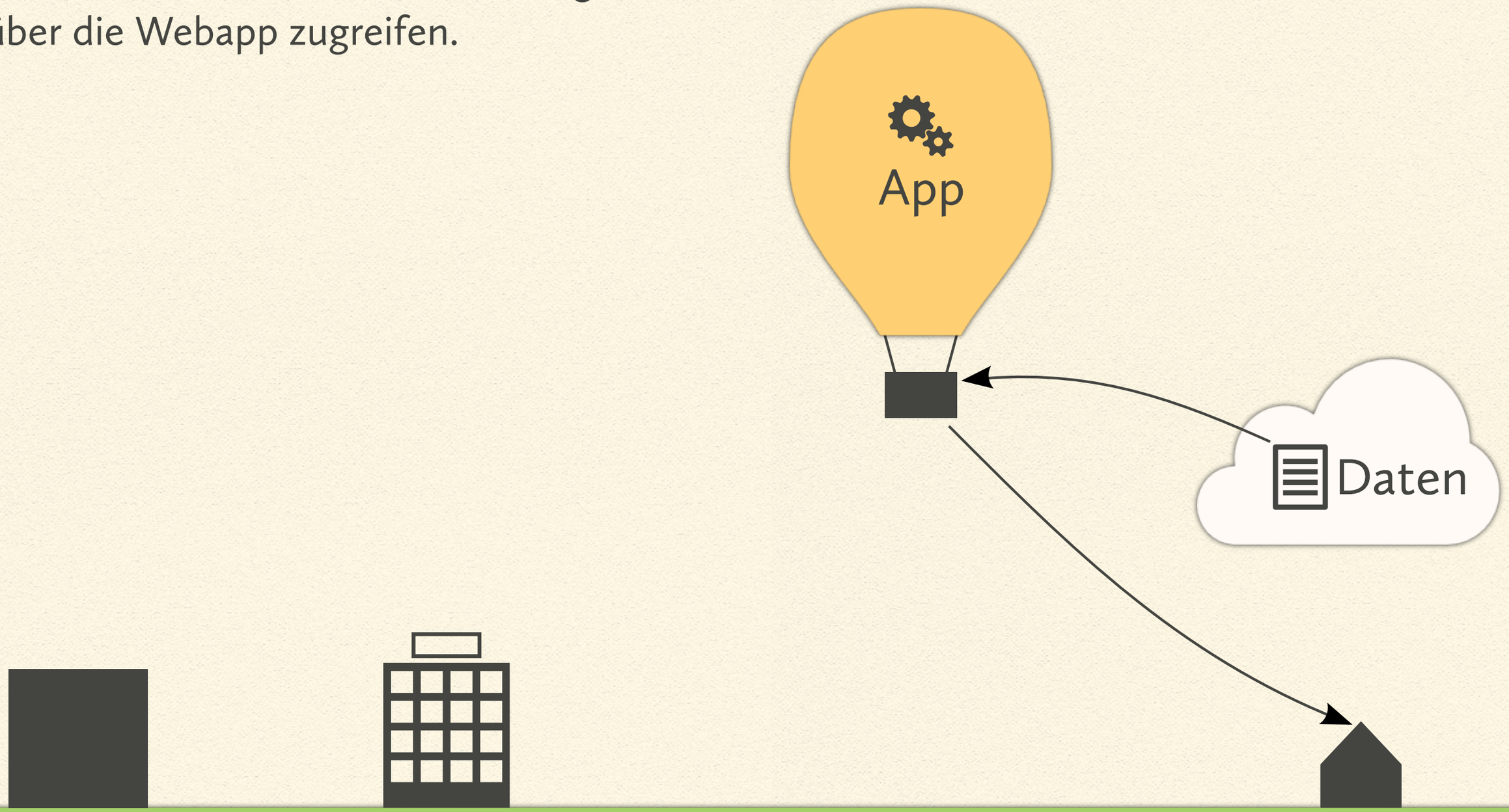


...aber Web-Entwickler würden weiterhin die Webapp zur Verfügung stellen.

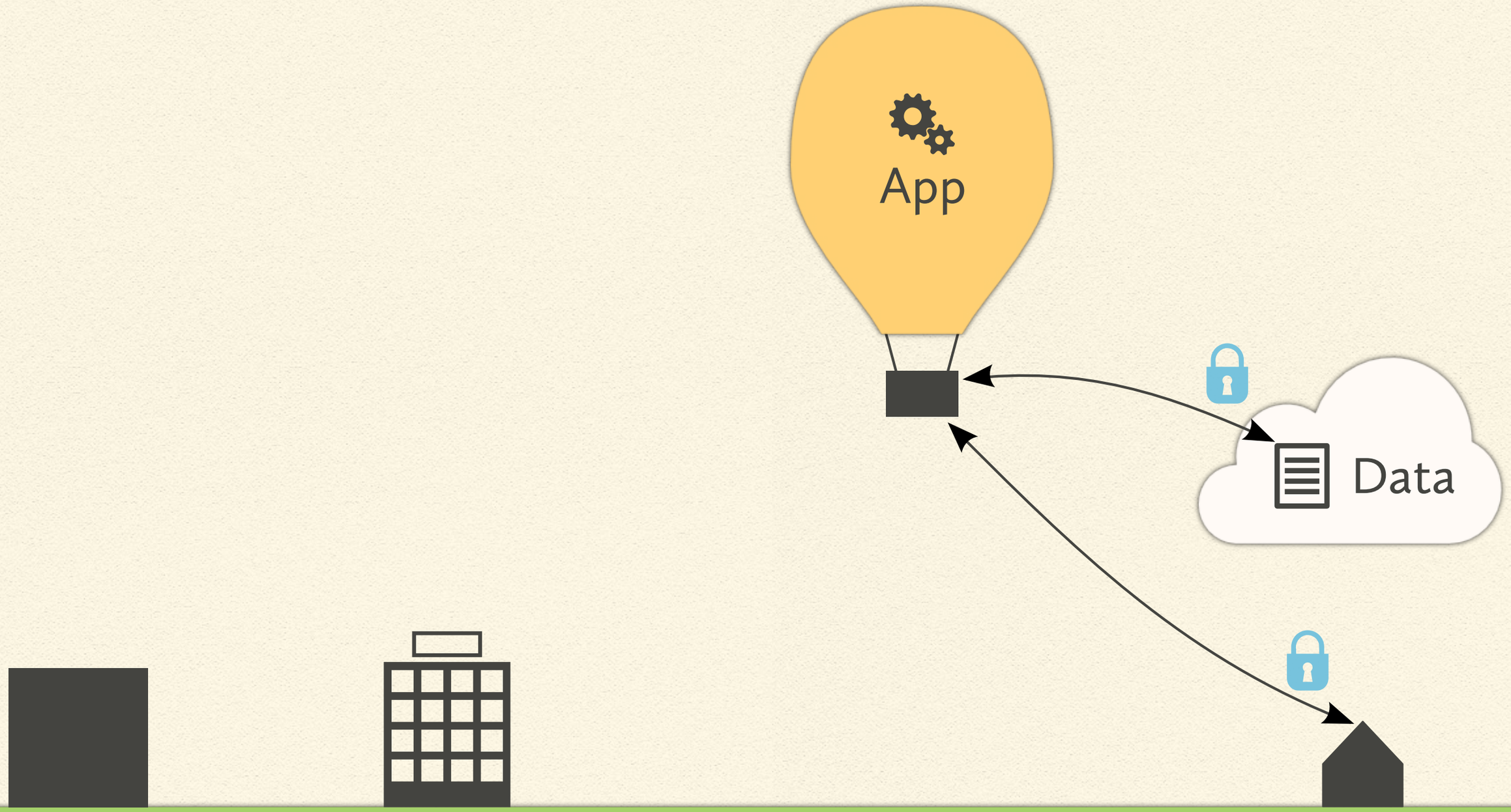
Sie würden die App nutzen, um Daten in ihrer eigenen Cloud zu speichern...



...und könnten auf ihre Daten wie gewohnt über die Webapp zugreifen.



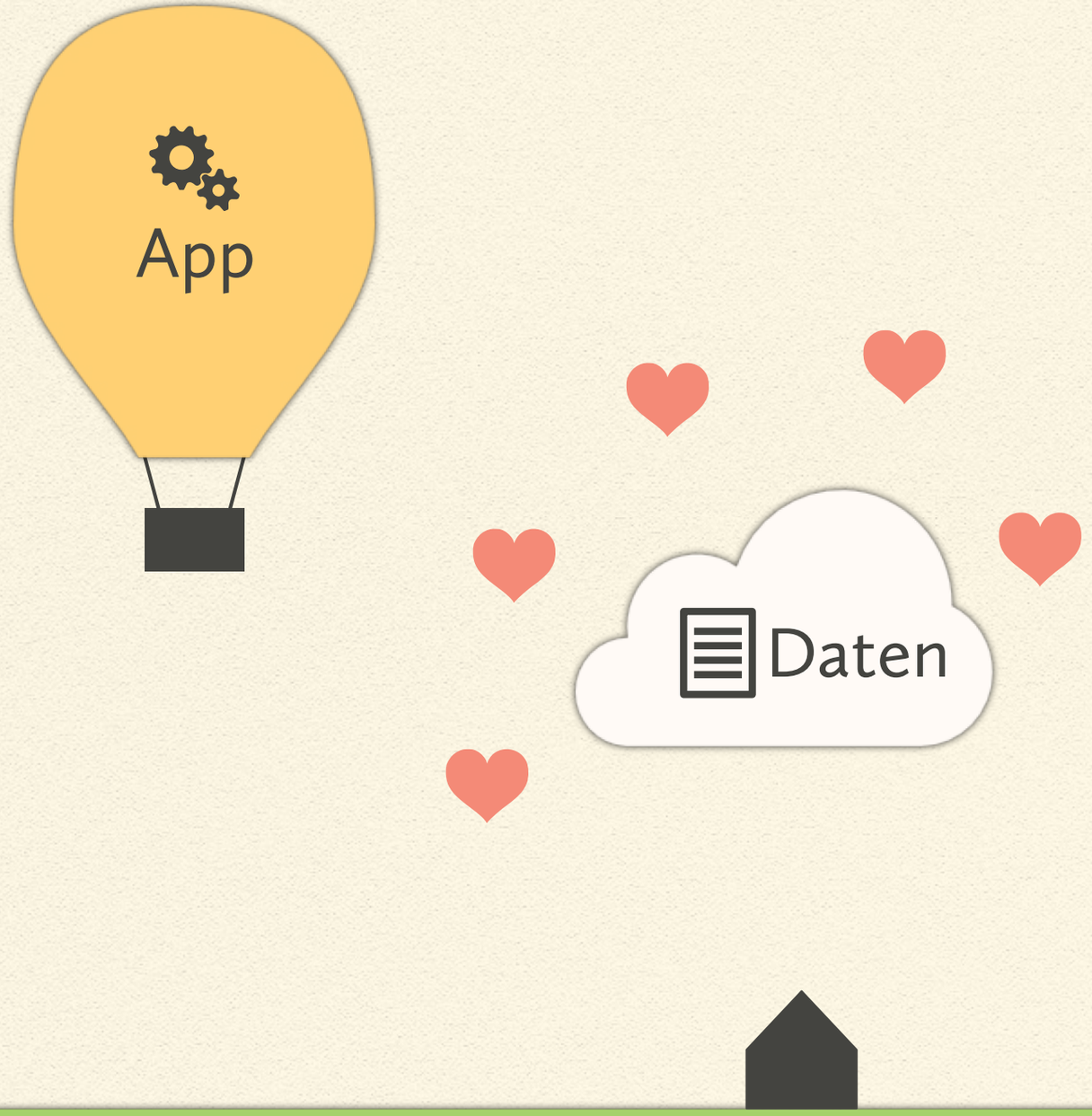
Alles wird auf ihrer Seite verschlüsselt...



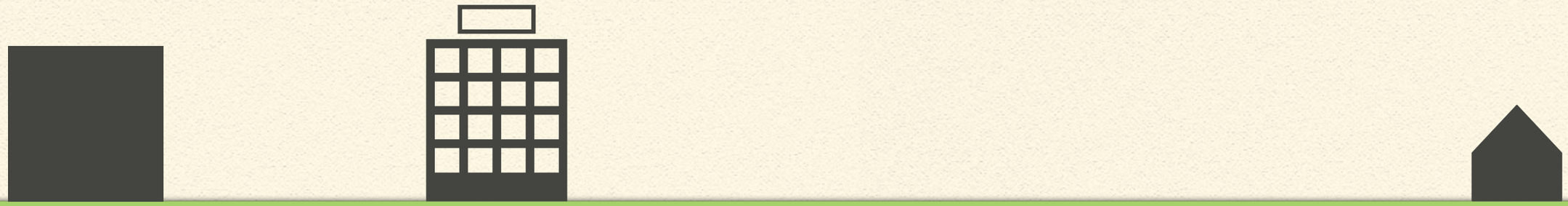
...so können weder der Webapp-Anbieter noch andere Gruppen ihre Daten lesen.

So haben Sie die einfache Handhabung einer Webapp...

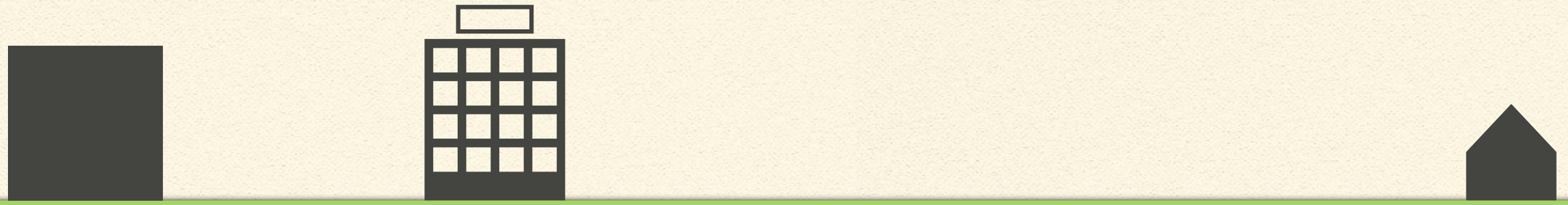
...und die Sicherheit der lokalen Cloud.

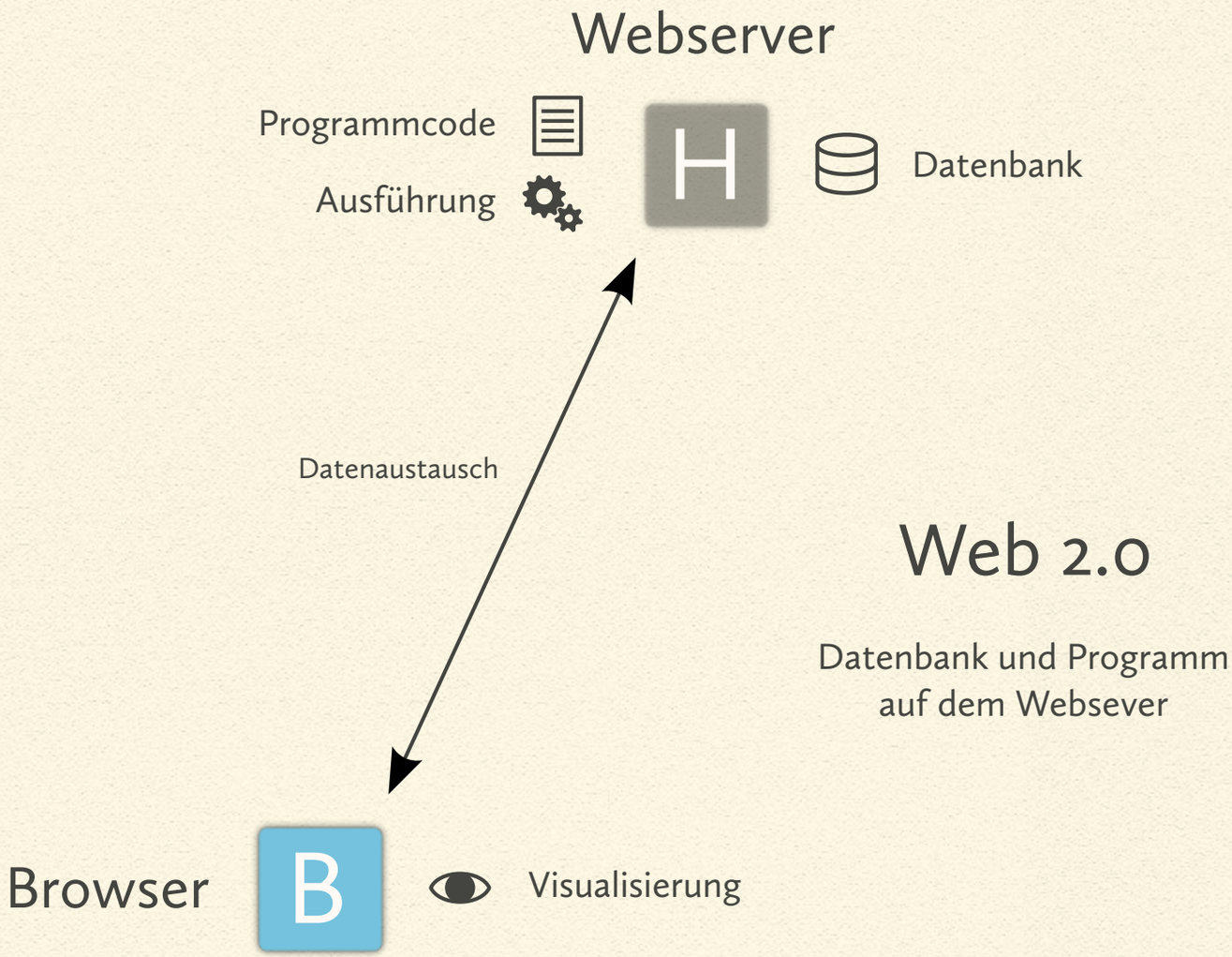


Darum geht es bei REDS.



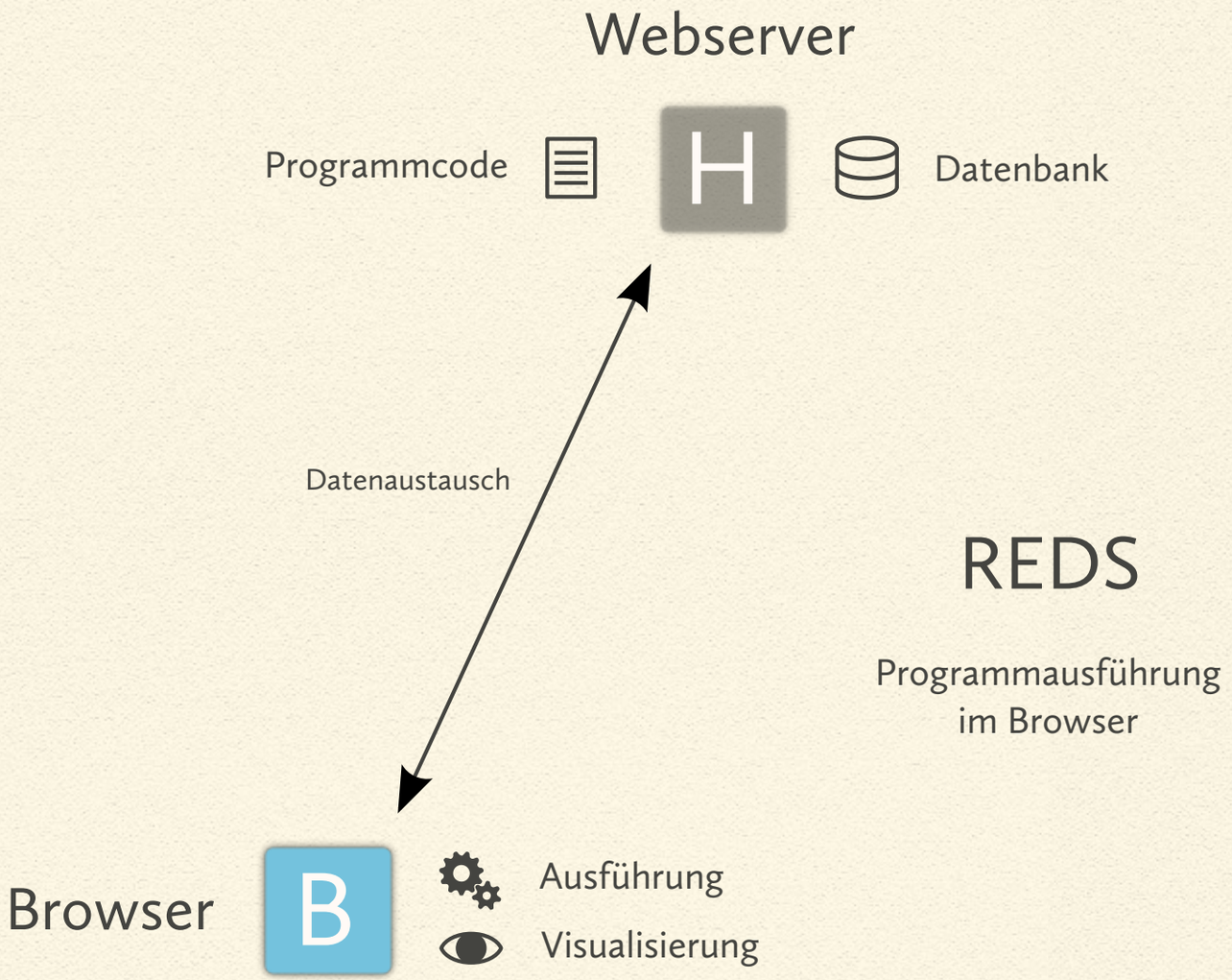
Aber wie funktioniert das genau?





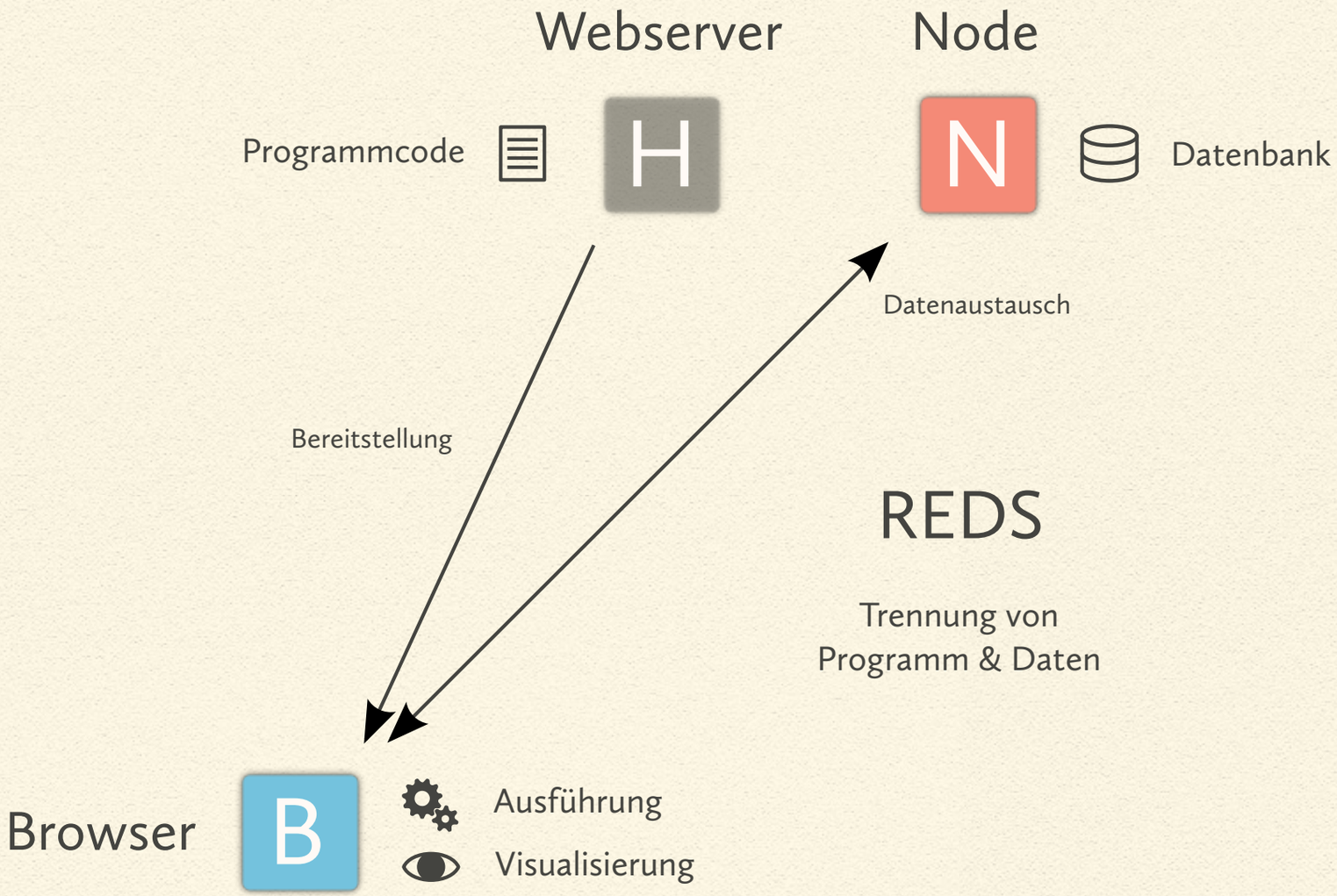
Web 2.0

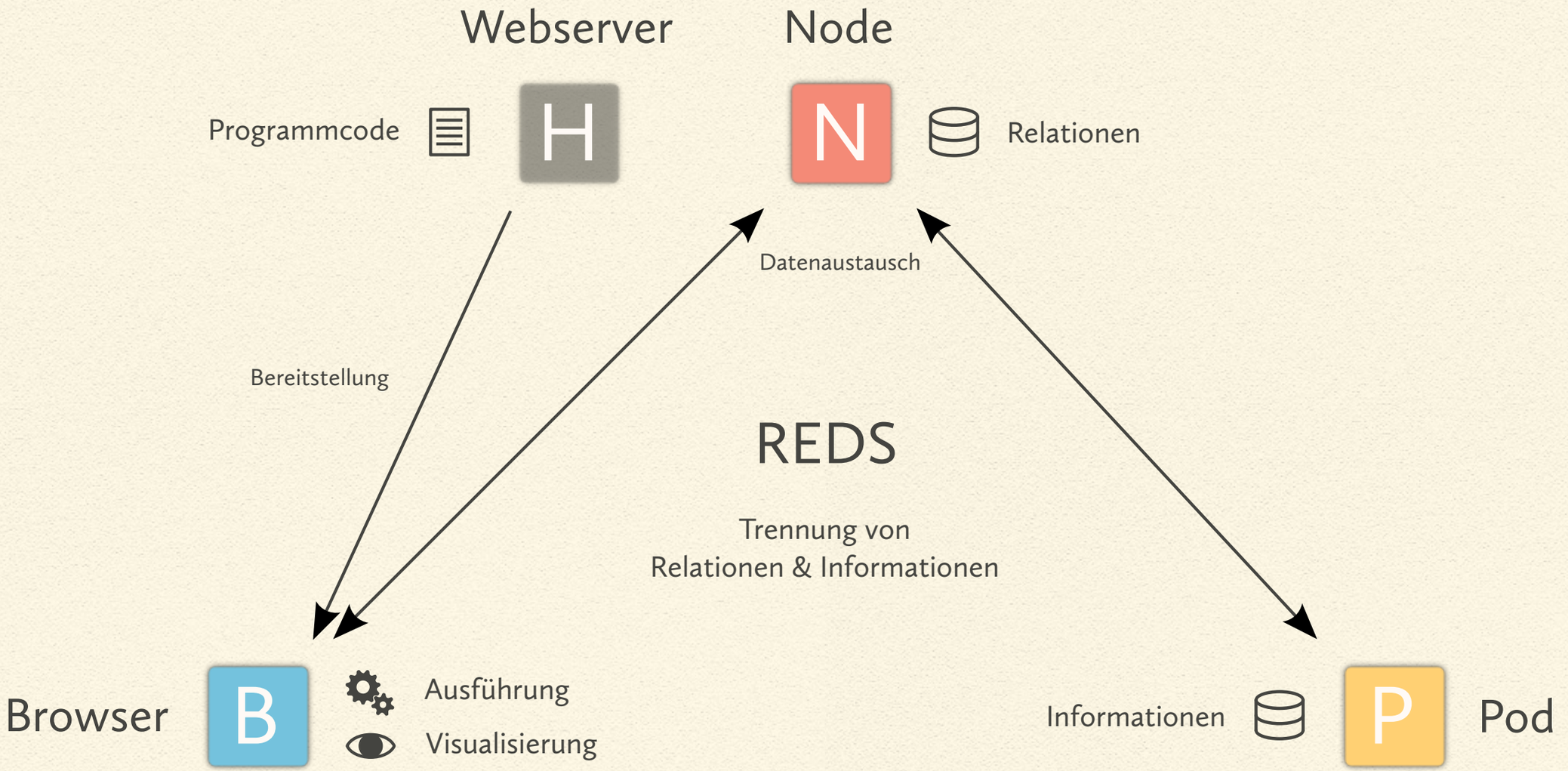
Datenbank und Programm
auf dem Websever



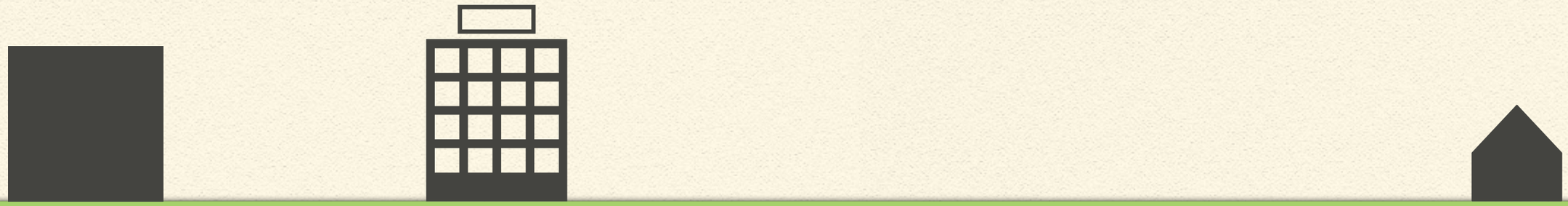
REDS

Programmausführung
im Browser


Webserver
Node
Programmcode
H
N
Datenbank
Datenaustausch
Bereitstellung
REDS
Trennung von
Programm & Daten
Browser
B
Ausführung
Visualisierung


Webserver
Node
Programmcode
H
N
Relationen
Datenaustausch
Bereitstellung
REDS
Trennung von
Relationen & Informationen
Browser
B
Ausführung
Visualisierung
Informationen
P
Pod

Welche Aufgaben haben die Komponenten?



REDS verteilt sich auf drei Komponenten:

Node

Browser

Pod

Der Anbieter kontrolliert den Node.
Der Nutzer Browser & Pod.

Node

Anbieter

Nutzer

Browser

Pod

Node

Browser

Verarbeitet Datensätze
(fat client)

Pod

Node

Verwaltet Datensätze
(speichert Relationen)

Browser

Verarbeitet Datensätze
(fat client)

Pod

Node

Verwaltet Datensätze
(speichert Relationen)

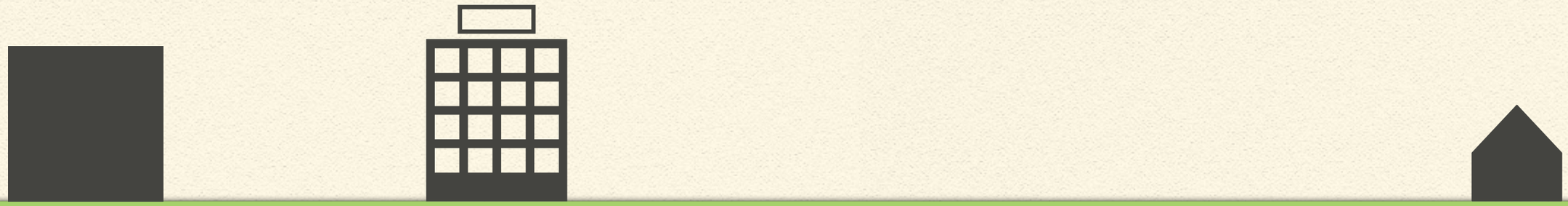
Browser

Verarbeitet Datensätze
(fat client)

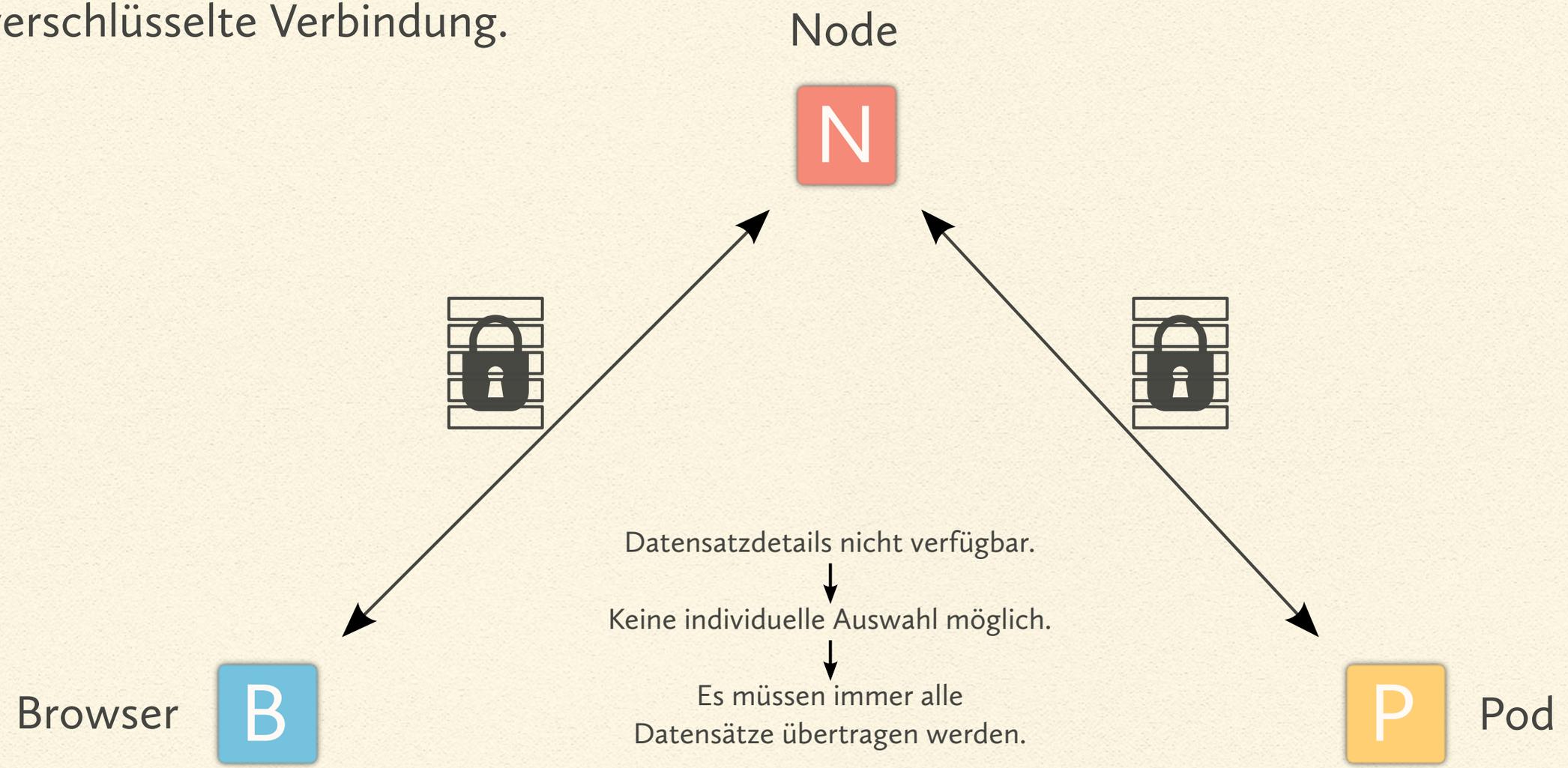
Speichert Datensätze
(unverschlüsselt)

Pod

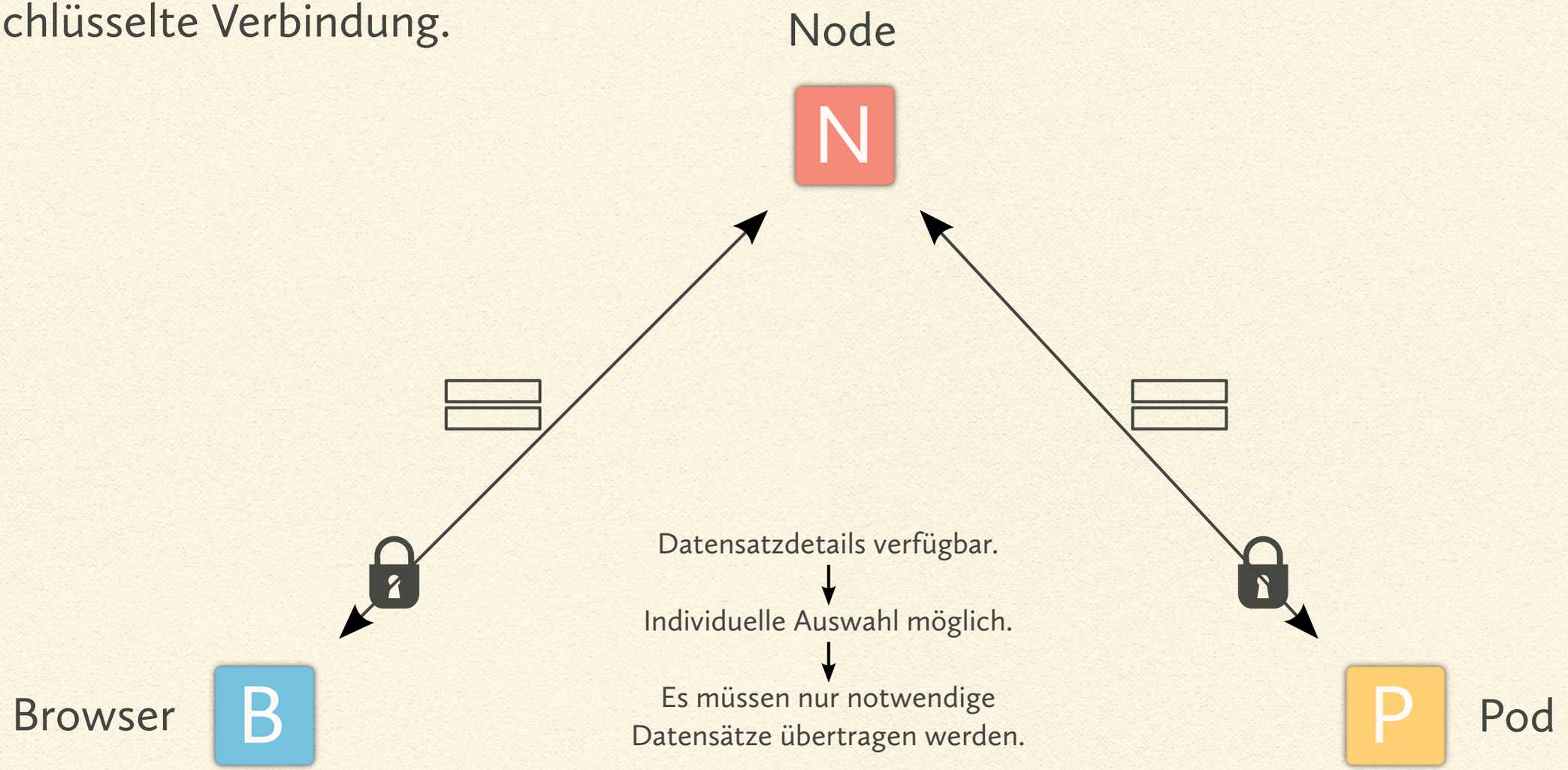
Warum werden die Daten unverschlüsselt gespeichert?



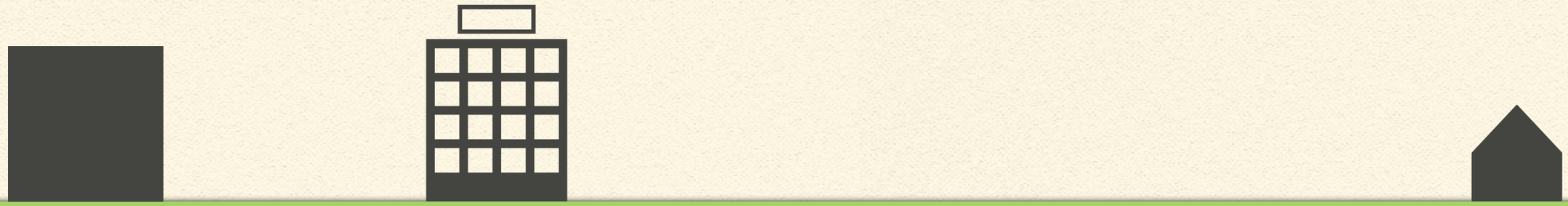
Verschlüsselte Daten.
Unverschlüsselte Verbindung.

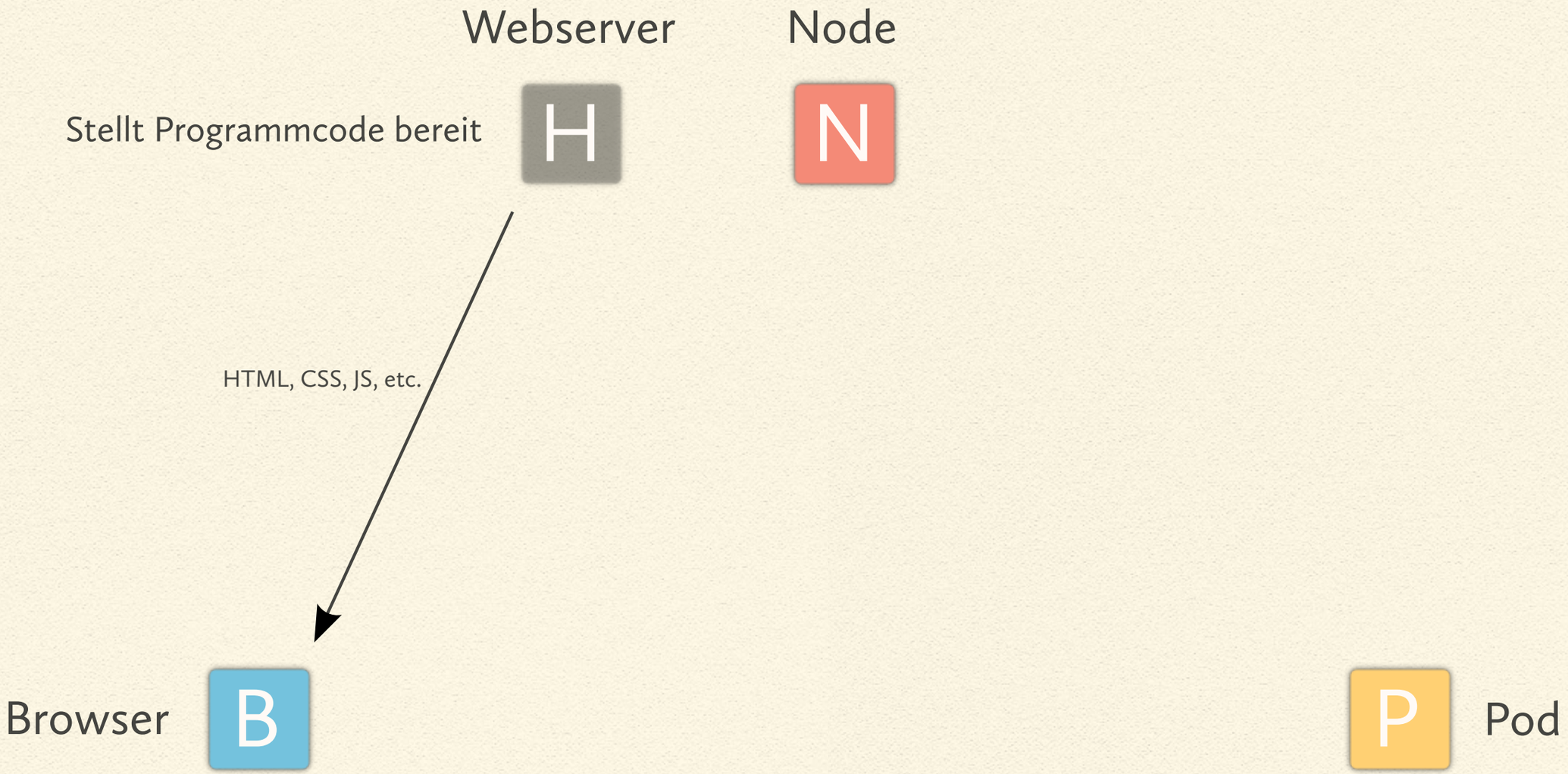


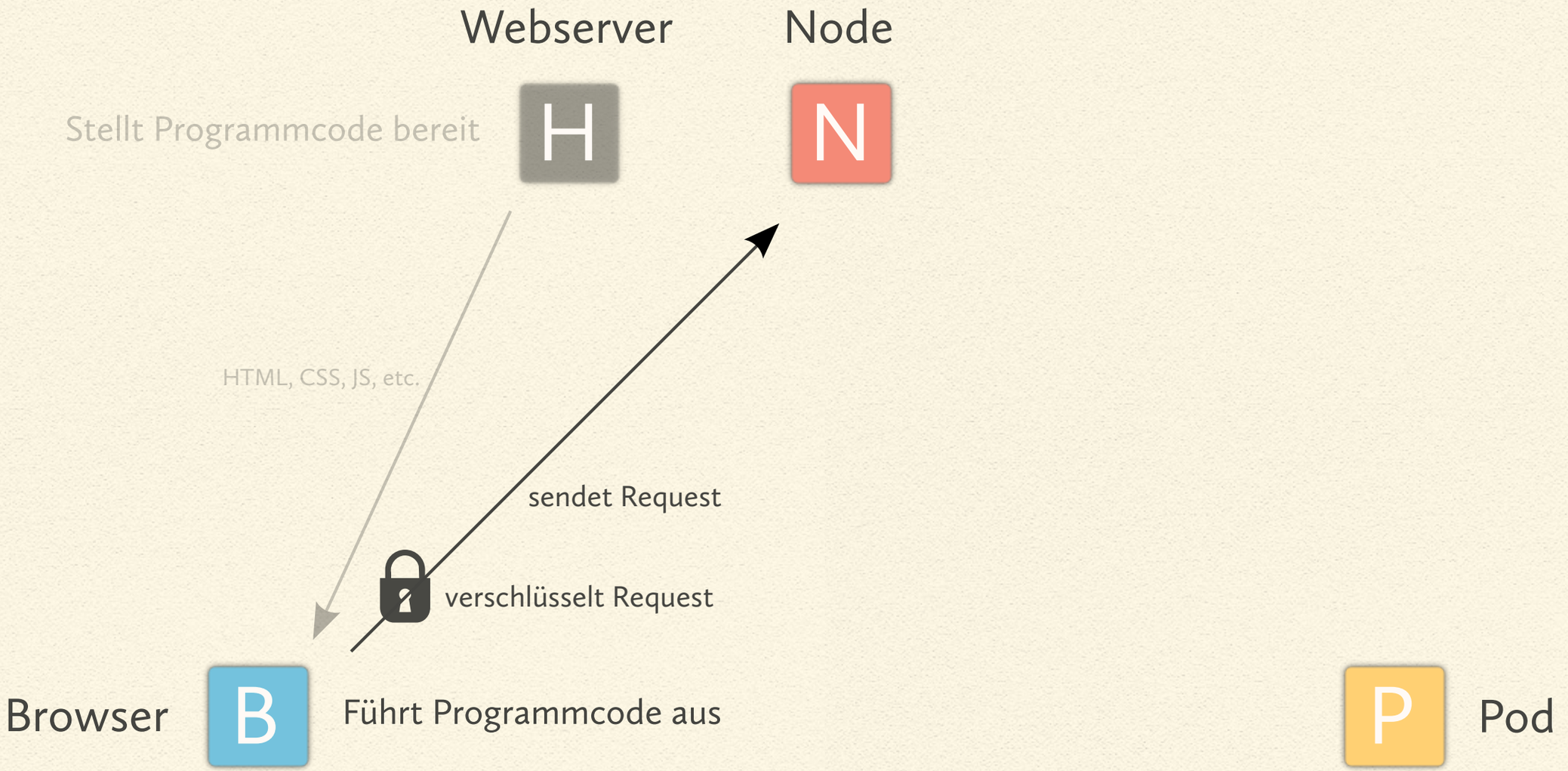
Unverschlüsselte Daten.
Verschlüsselte Verbindung.

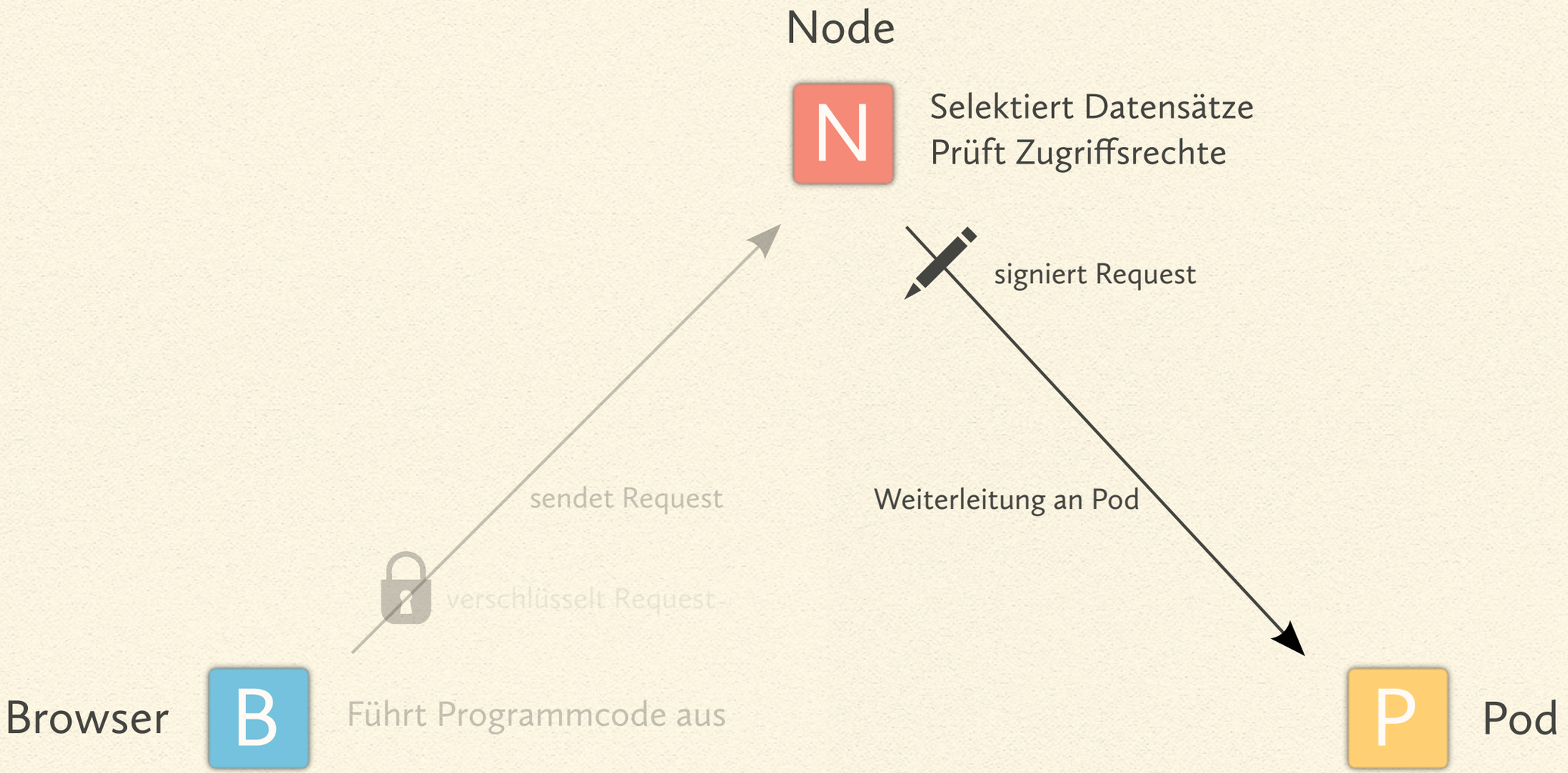


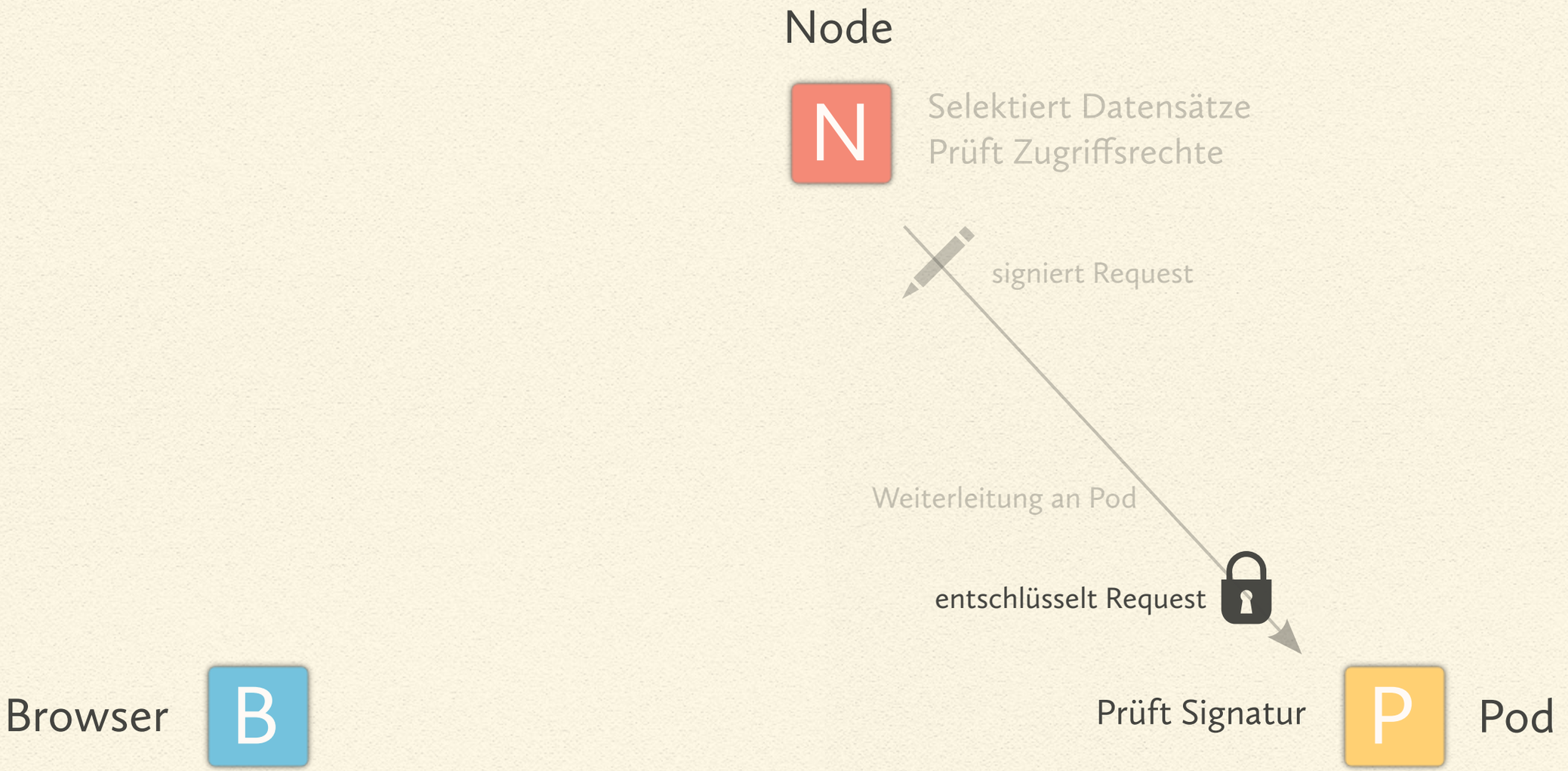
Wie arbeiten die Komponenten zusammen?

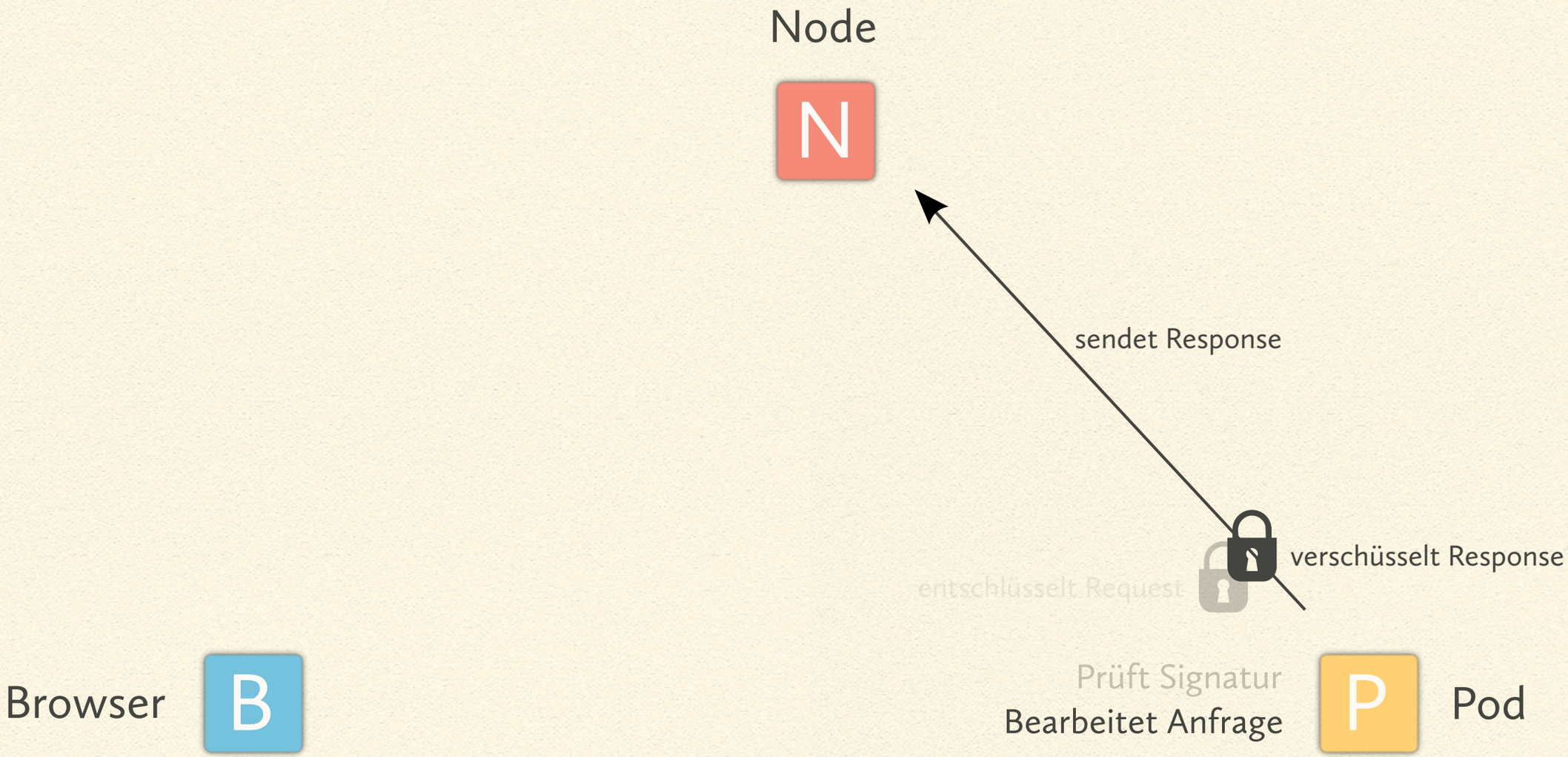


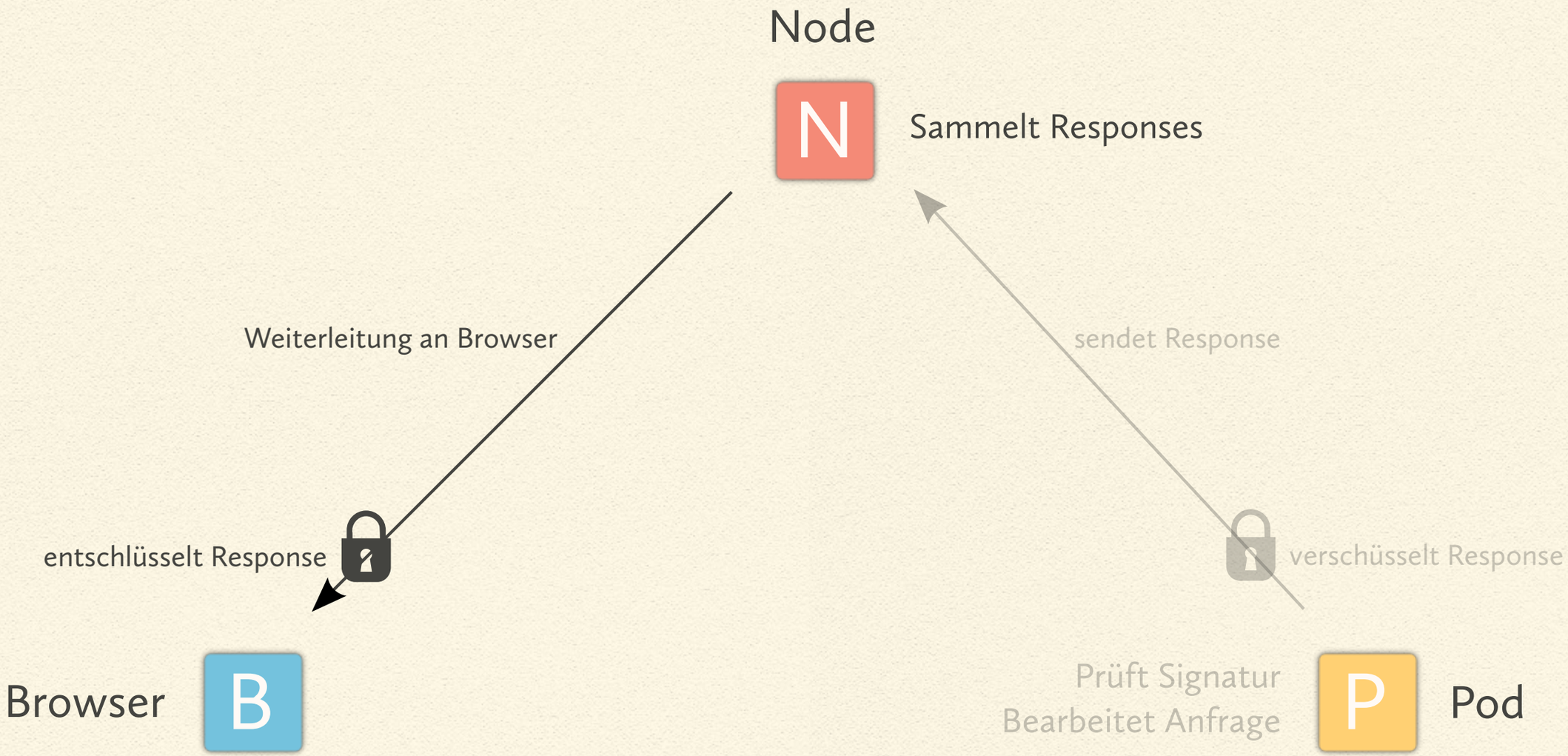

Webserver
Node
Stellt Programmcode bereit
H
N
HTML, CSS, JS, etc.
Browser
B
P
Pod

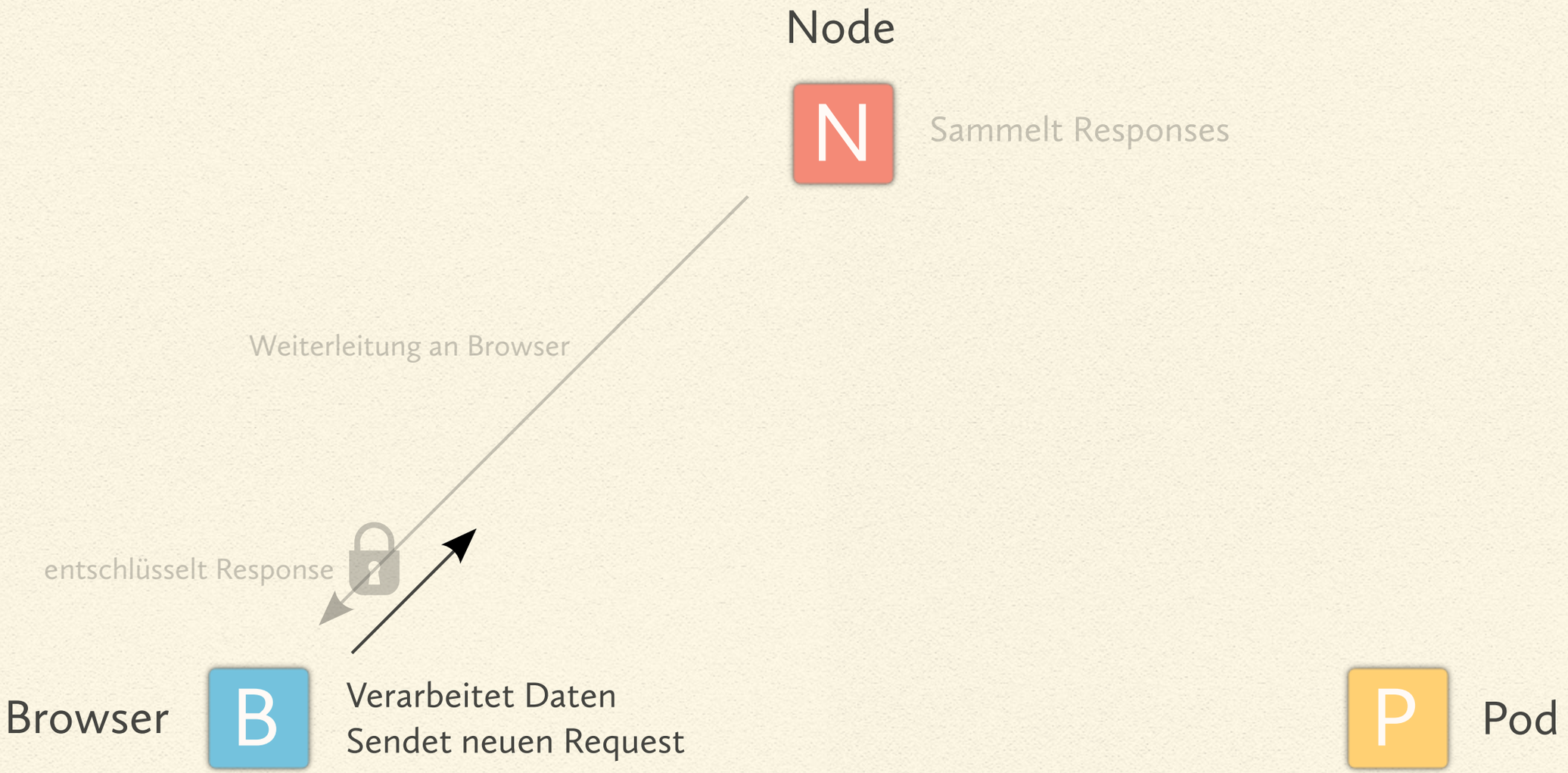

Webserver
Node
Stellt Programmcode bereit
H
N
HTML, CSS, JS, etc.
sendet Request
verschlüsselt Request
Browser
B
Führt Programmcode aus
P
Pod

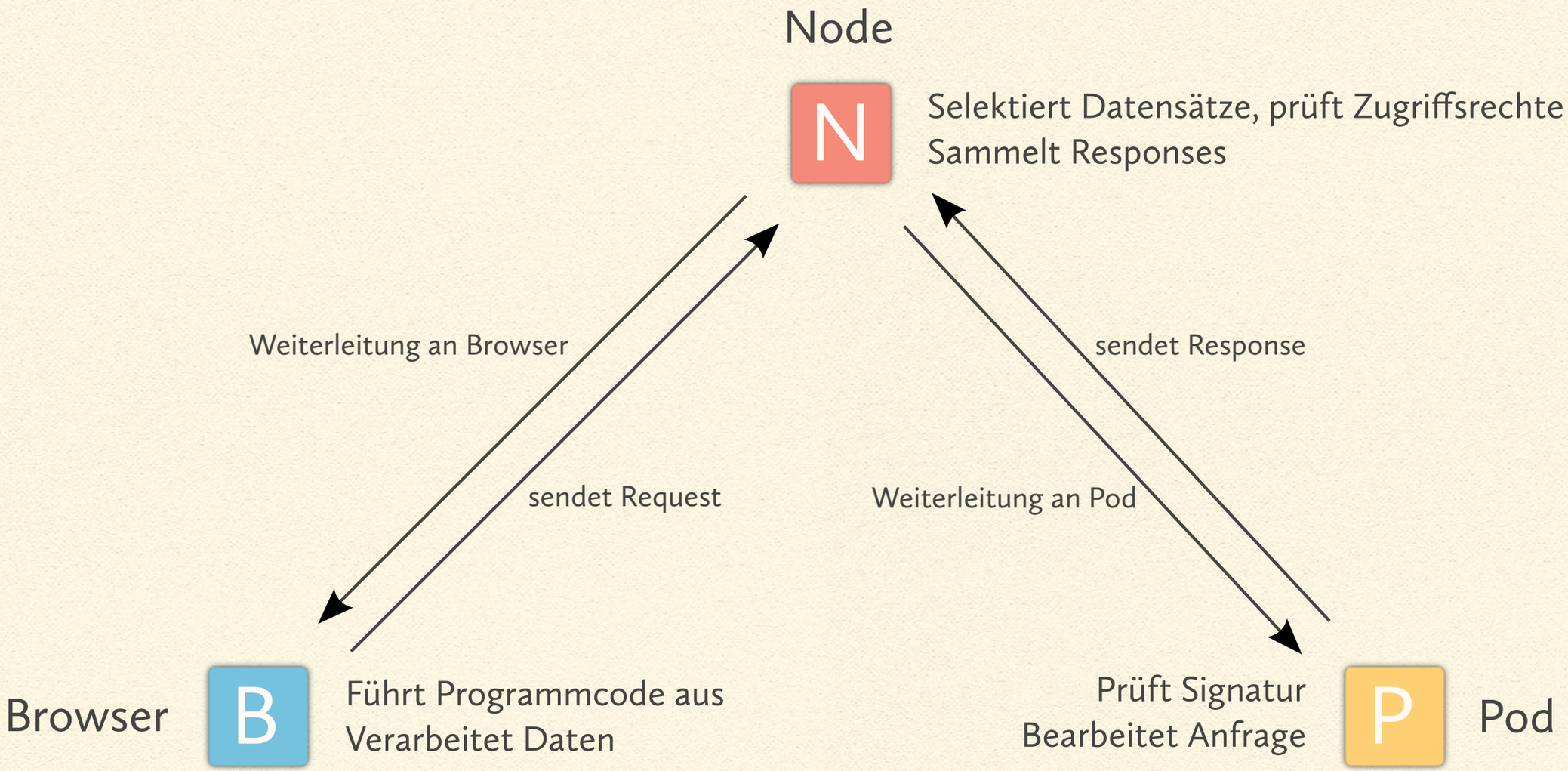

Node
N
Selektiert Datensätze
Prüft Zugriffsrechte
signiert Request
sendet Request
Weiterleitung an Pod
verschlüsselt Request
Browser
B
Führt Programmcode aus
P
Pod


Node
N
Selektiert Datensätze
Prüft Zugriffsrechte
signiert Request
Weiterleitung an Pod
entschlüsselt Request
Browser
B
Prüft Signatur
P
Pod

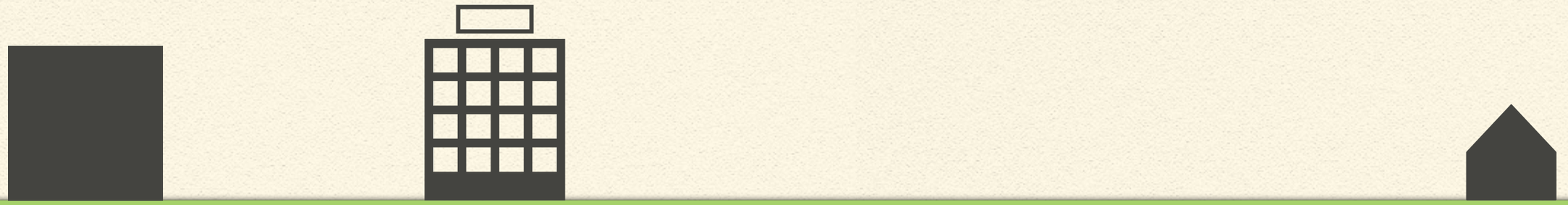

Node
N
sendet Response
verschüsselt Response
entschlüsselt Request
Prüft Signatur
Bearbeitet Anfrage
P
Pod
Browser
B


Node
N
Sammelt Responses
Weiterleitung an Browser
sendet Response
entschlüsselt Response
verschüsselt Response
Browser
B
Prüft Signatur
Bearbeitet Anfrage
P
Pod


Node
N
Sammelt Responses
Weiterleitung an Browser
entschlüsselt Response
Browser
B
Verarbeitet Daten
Sendet neuen Request
P
Pod


Node
N
Selektiert Datensätze, prüft Zugriffsrechte
Sammelt Responses
Weiterleitung an Browser
sendet Request
sendet Response
Weiterleitung an Pod
Browser
B
Führt Programmcode aus
Verarbeitet Daten
Prüft Signatur
Bearbeitet Anfrage
P
Pod

Welche Kompenenten besitzen welche Informationen?



Node

Anbieter

Nutzer

Browser

Name, Password, Pseudonym
Authentifizierungs-Schlüssel
Datensatz-Schlüssel
Datensatz (zur Verarbeitung)

Pod

Node

Pseudonym
Authentifizierungs-Schlüssel
Signierungs-Schlüssel

Anbieter

Nutzer

Browser

Name, Password, Pseudonym
Authentifizierungs-Schlüssel
Datensatz-Schlüssel
Datensatz (zur Verarbeitung)

Pod

Node

Pseudonym
Authentifizierungs-Schlüssel
Signierungs-Schlüssel

Anbieter

Nutzer

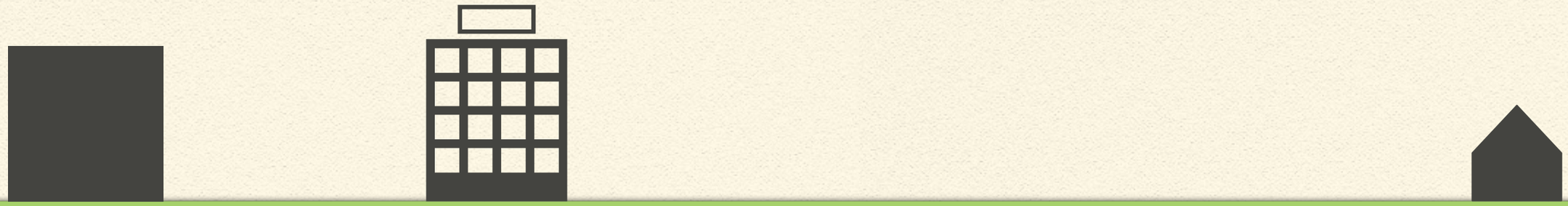
Browser

Name, Password, Pseudonym
Authentifizierungs-Schlüssel
Datensatz-Schlüssel
Datensatz (zur Verarbeitung)

Signierungs-Schlüssel
Datensatz-Schlüssel
Datensatz (zur Speicherung)

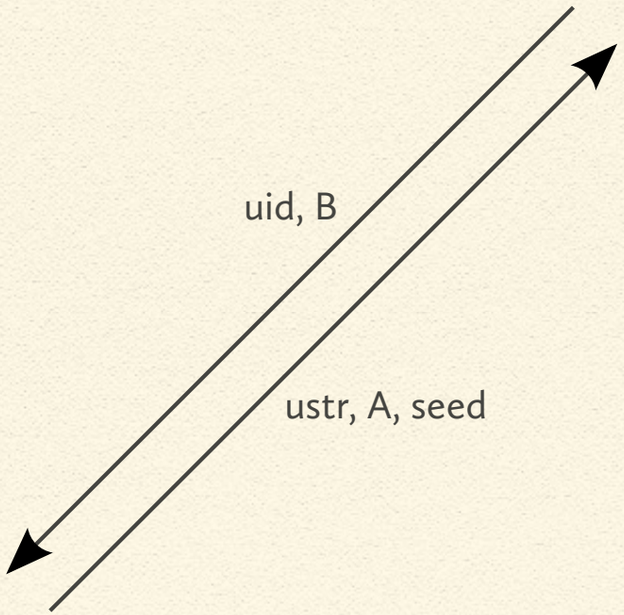
Pod

Wie funktioniert die Kommunikation im Detail?



Nutzer registrieren

POST /!/user



N

b = rand()
b => B
akey = dh(A, b)
ustr, B, akey ▸ DB[uid]

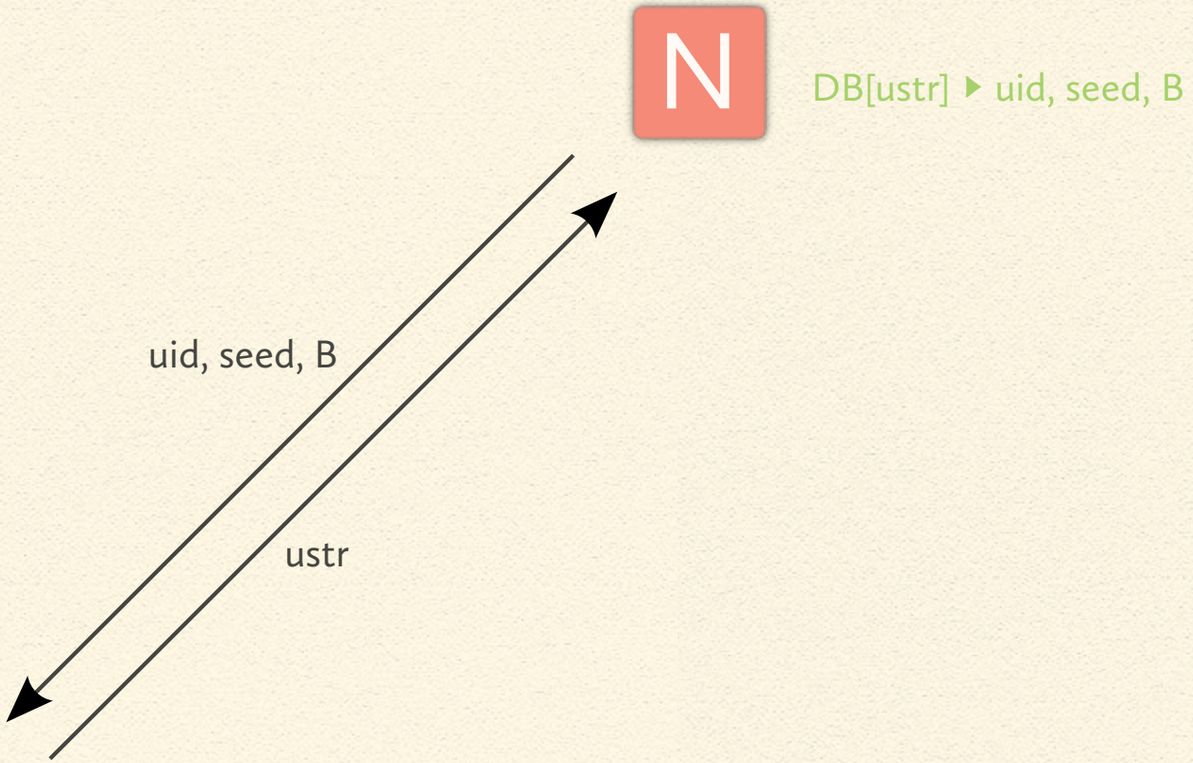
USER ▸ name, pw
seed = rand()
ustr = hash(pw.name)
a = hash(pw.seed)
a => A
--------- Kommunikation mit Node ---------
akey = dh(a, B)
uid, akey ▸ RAM

B

P

Nutzer anmelden

GET /!/user/name



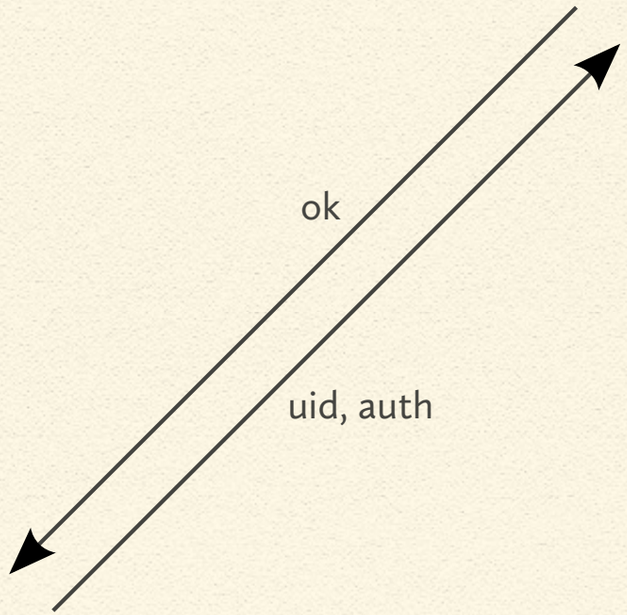
USER ▸ name, pw
ustr = hash(pw.seed)
--------- Kommunikation mit Node ---------
a = hash(pw.seed)
akey = dh(a, B)
uid, akey ▸ RAM

Nutzer authentifizieren

GET /!/authentification (und andere)

DB[uid] ▸ akey
m' = method'.url'.data'
auth' = hmac(m', akey)
auth == auth' ?

N



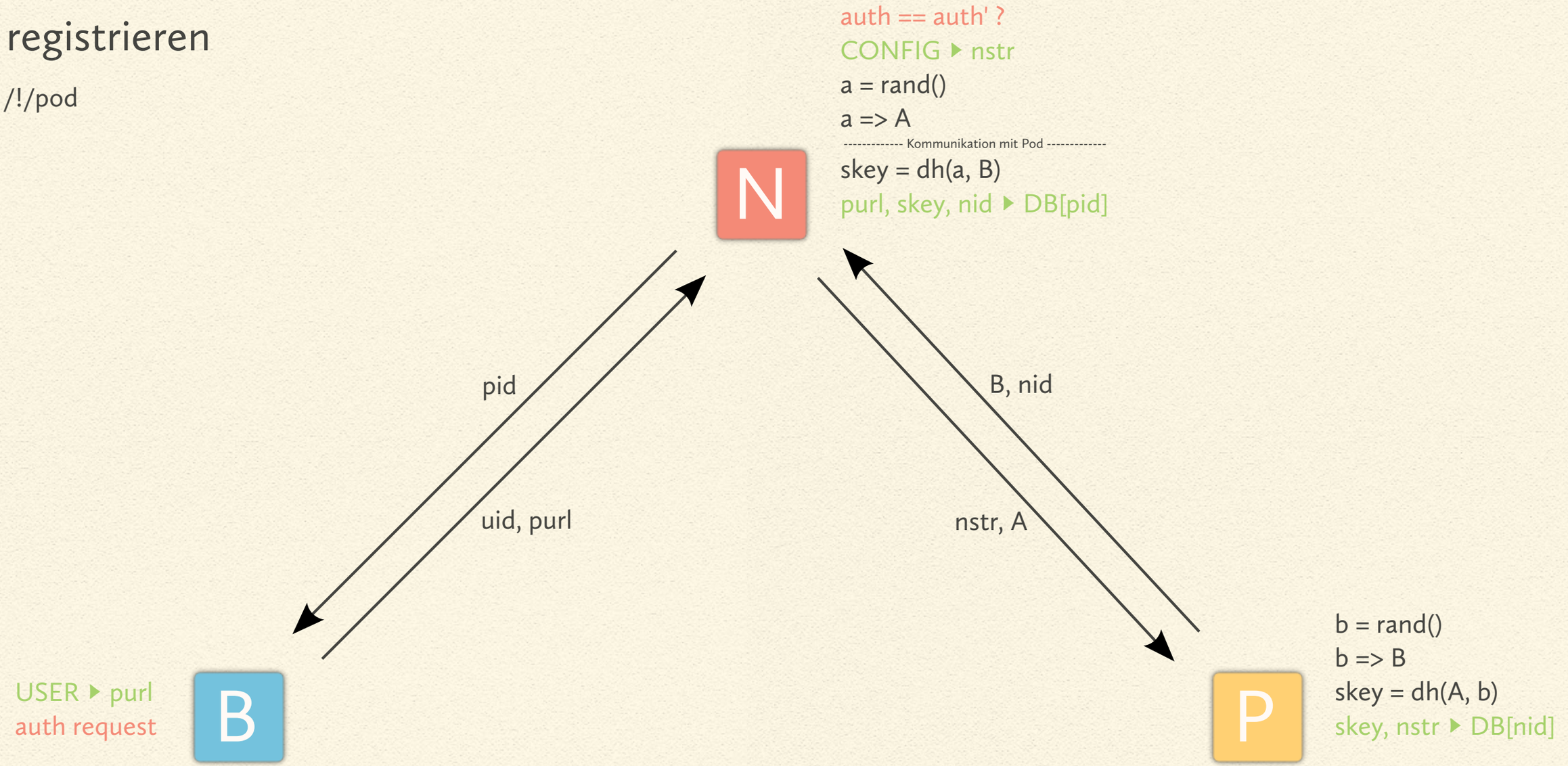
RAM ▸ uid, akey
m = method.url.data
auth = hmac(m, akey)

B

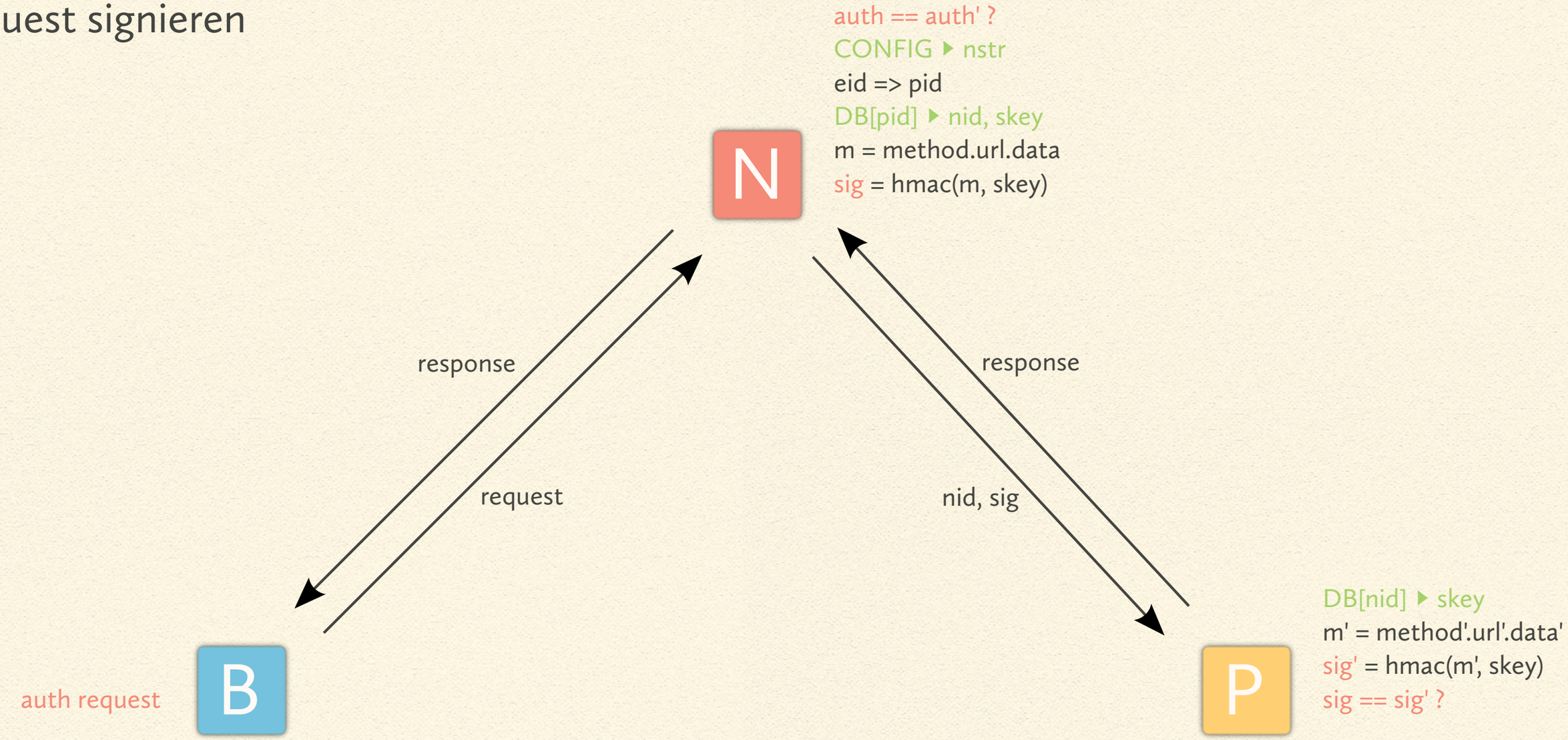
P

Pod registrieren

POST /!/pod

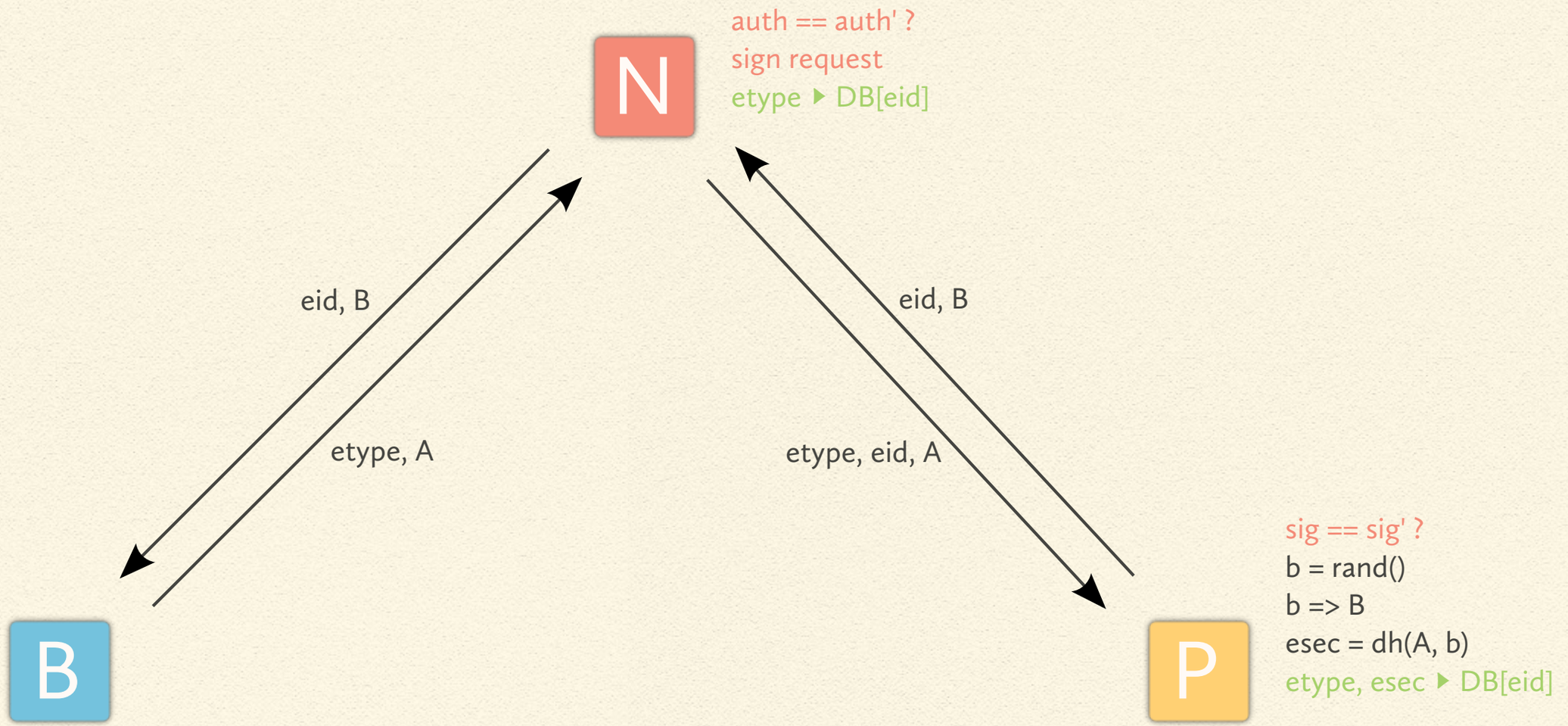


Request signieren



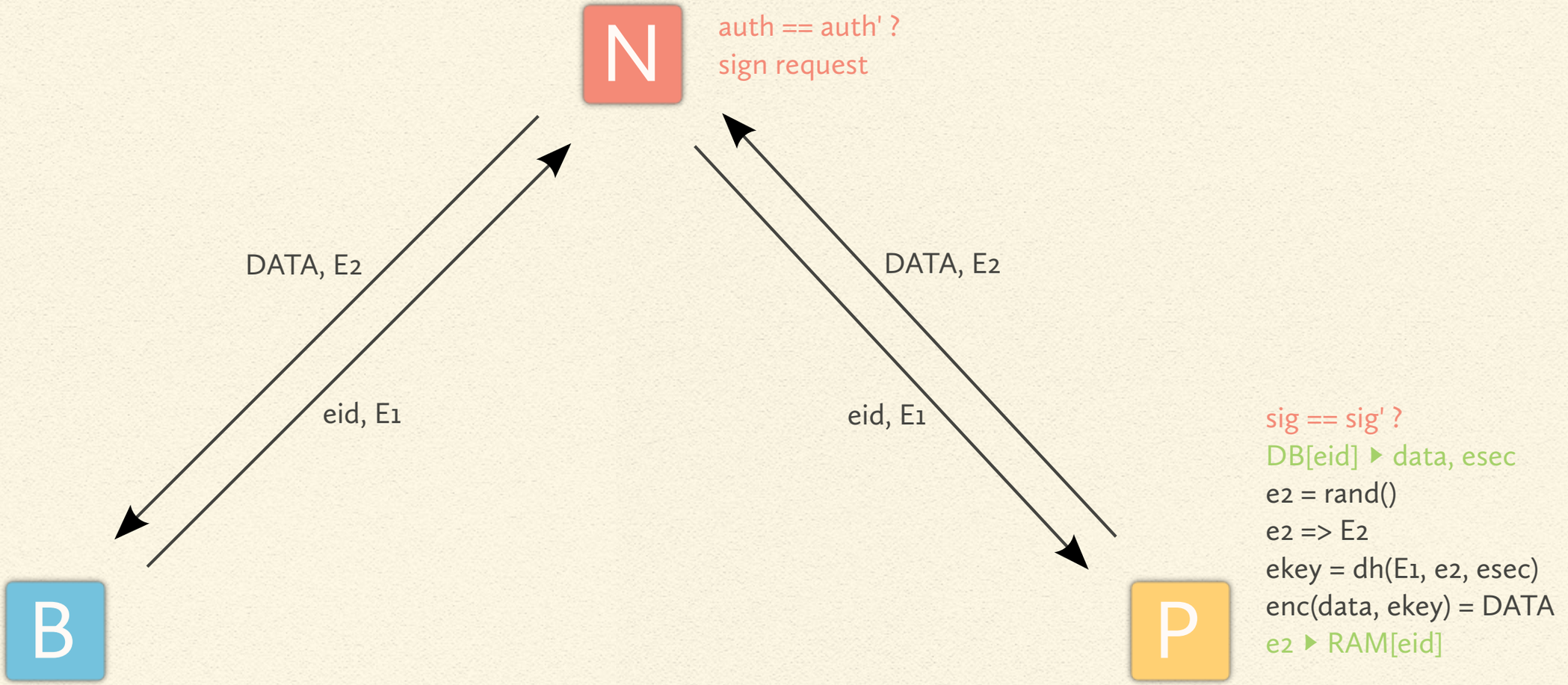
Entity erstellen

POST /etype



Entity lesen

GET /etype/eid

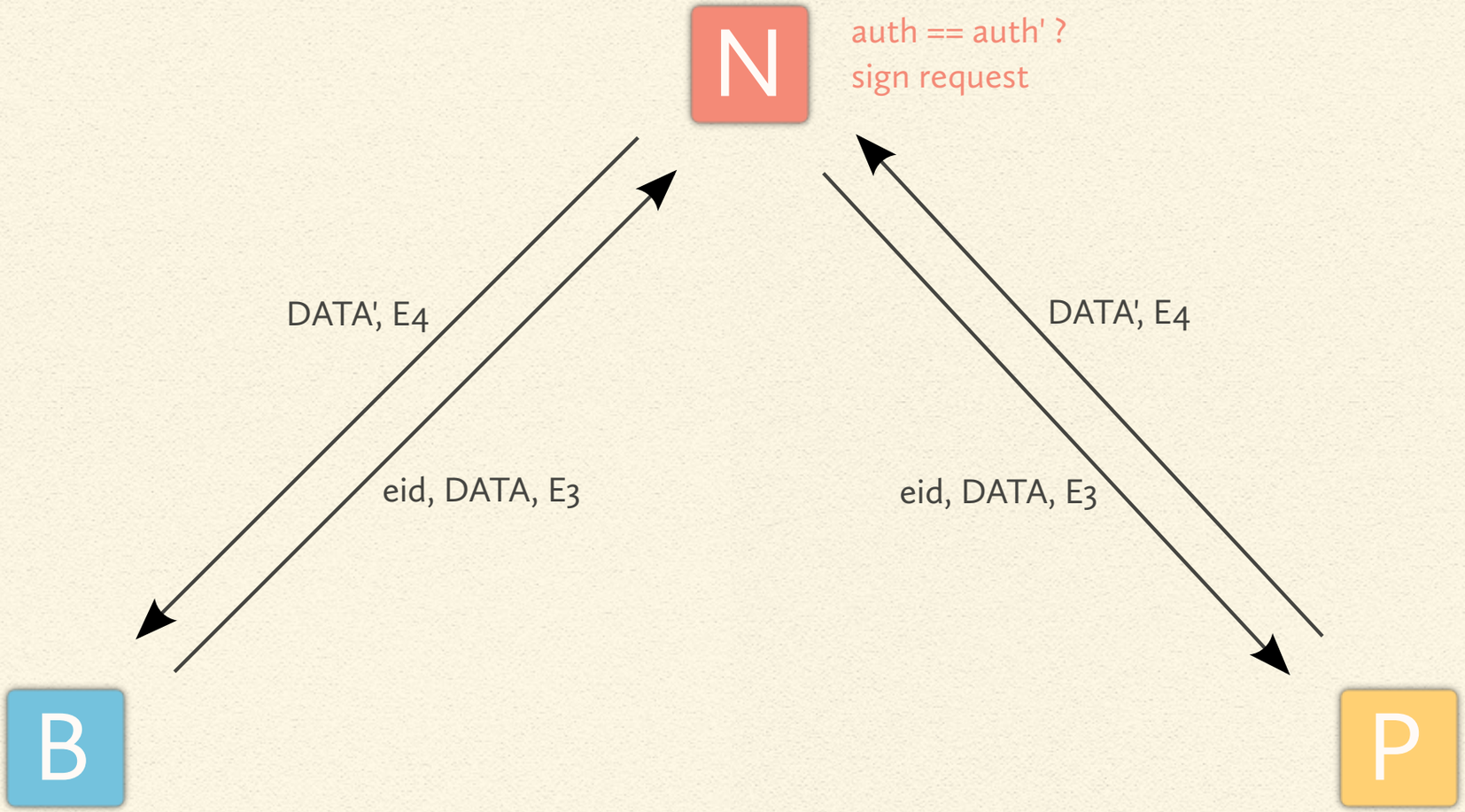


USER ▸ eid
RAM[eid] ▸ esec
e1 = rand()
e1 => E1
auth request
---------- Kommunikation mit Node ----------
ekey = dh(e1, E2, esec)
dec(DATA, ekey) = data
E2 ▸ RAM[eid]

sig == sig' ?
DB[eid] ▸ data, esec
e2 = rand()
e2 => E2
ekey = dh(E1, e2, esec)
enc(data, ekey) = DATA
e2 ▸ RAM[eid]

Entity ändern

PUT /etype/eid

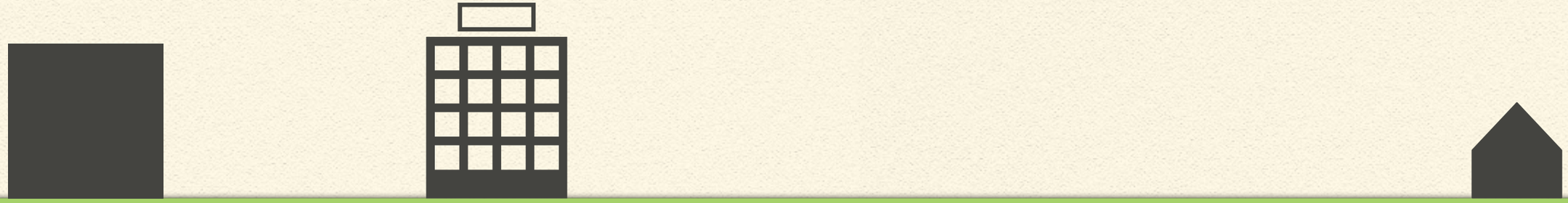


USER ▸ eid
RAM[eid] ▸ esec, E2
e3 = rand()
e3 => E3
ekey = dh(E2, e3, esec)
DATA = enc(data, ekey)
auth request
---------- Kommunikation mit Node ----------
ekey' = dh(e3, E4, esec)
data' = dec(DATA', ekey')
E2 ▸ RAM[eid]

sig == sig' ?
RAM[eid] ▸ esec, e2
ekey = dh(e2, E3, esec)
data = dec(DATA, ekey)
---------- Verarbeiten der Anfrage ----------
e4 = rand()
e4 => E4
ekey' = dh(E3, e4, esec)
DATA' = enc(data', ekey')
e4 ▸ RAM[eid]

REDS wird im Frühjar 2014 als

Open-Source verfügbar sein.



REDS wird entwickelt von:

Flowy Apps

Mehr Informationen finden Sie auf http://flowyapps.com