# OBJEKTE & STATE

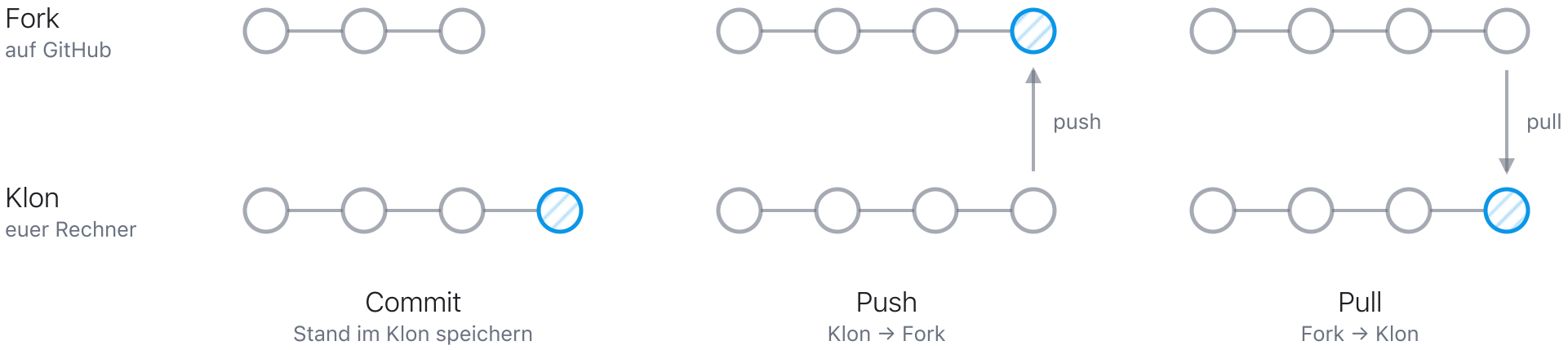
Michael Kluge & Fabian Rafreider

# UNSER AKTUELLER STAND

- **Original** — unser Repo auf GitHub
- **Fork** — eure Kopie davon auf GitHub
- **Klon** — der Ordner auf eurem Rechner

Gerade sind die drei wahrscheinlich **nicht** gleich.

# KLON UND FORK ABGLEICHEN



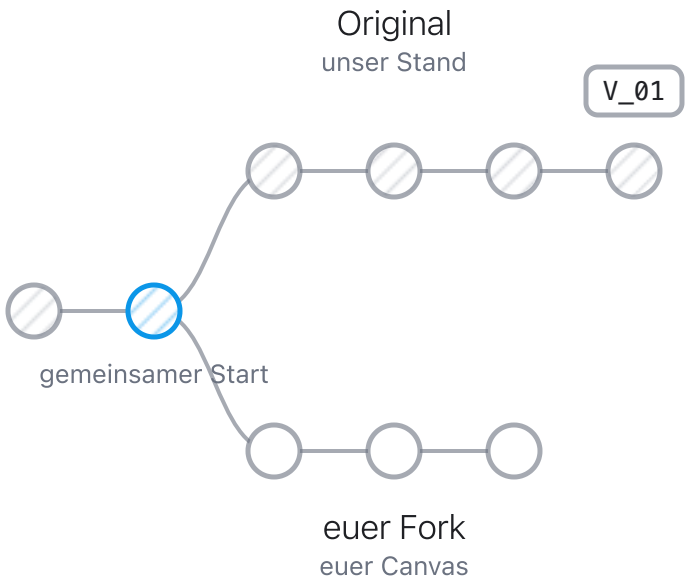

Jeder Commit baut auf dem vorigen auf. Die Reihe der Commits ist ein **Branch** — unserer heißt `main`.

# ORIGINAL UND FORK LAUFEN AUSEINANDER

- Ihr habt vom Original geforkt und weitergebaut — euer Canvas
- Wir haben im Original parallel weitergebaut — unser Stand `V_01`
- Beide starten beim selben Commit und gehen danach getrennte Wege

`V_01` ist ein **Tag**: ein festes Etikett auf genau einem Commit.

Wie kommt unser Stand zu euch?

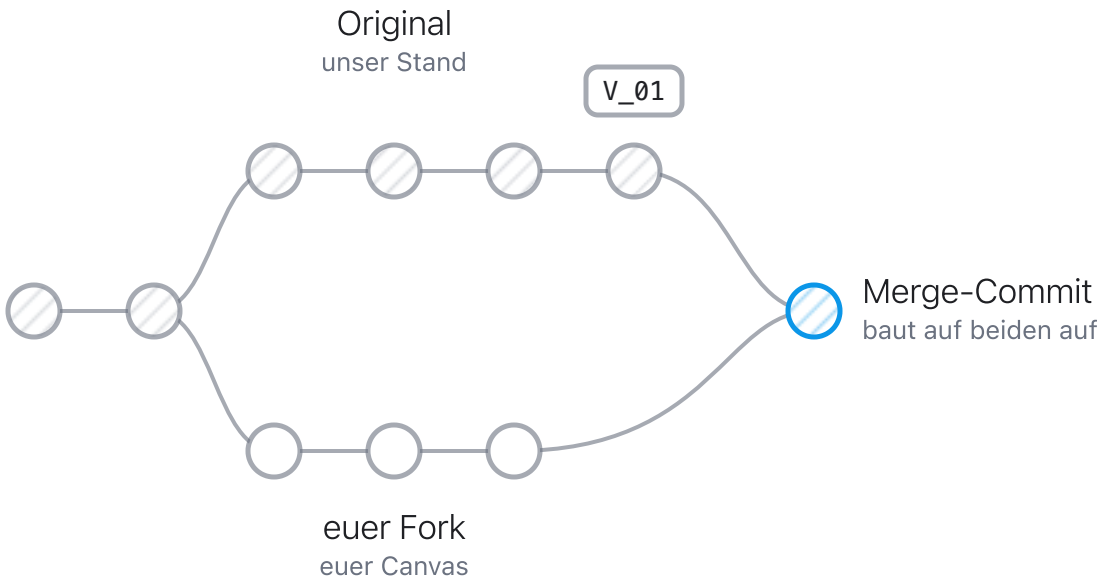



Gleicher Start, danach getrennte Wege.

# MERGE = ZUSAMMENFÜHREN

- Git vergleicht beide Stände, Datei für Datei, Zeile für Zeile
- Was nur **eine** Seite geändert hat, wird einfach übernommen
- Wo **beide** dieselbe Stelle geändert haben → **Konflikt**
- Bei einem Konflikt muss ein Mensch entscheiden, welche Fassung gilt

Das Ergebnis ist ein neuer Commit, der auf **beiden** aufbaut.



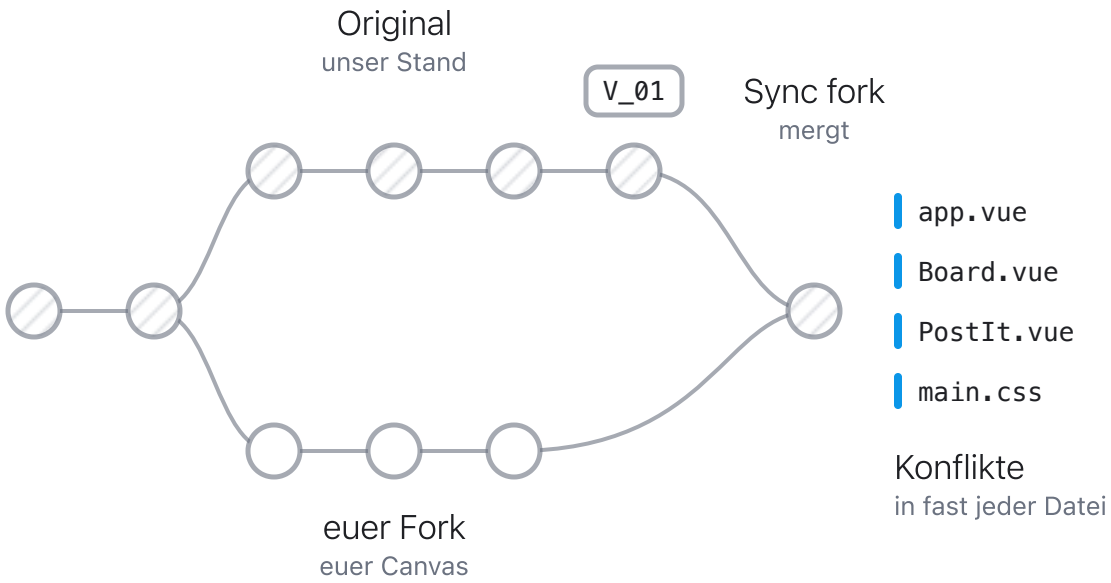

Ein Merge-Commit hat zwei Vorgänger — beide Geschichten bleiben.

# WARUM NICHT „SYNC FORK“?

GitHub zeigt in eurem Fork: euer Stand weicht vom Original ab — **Sync fork?**

Der Button **mergt**.

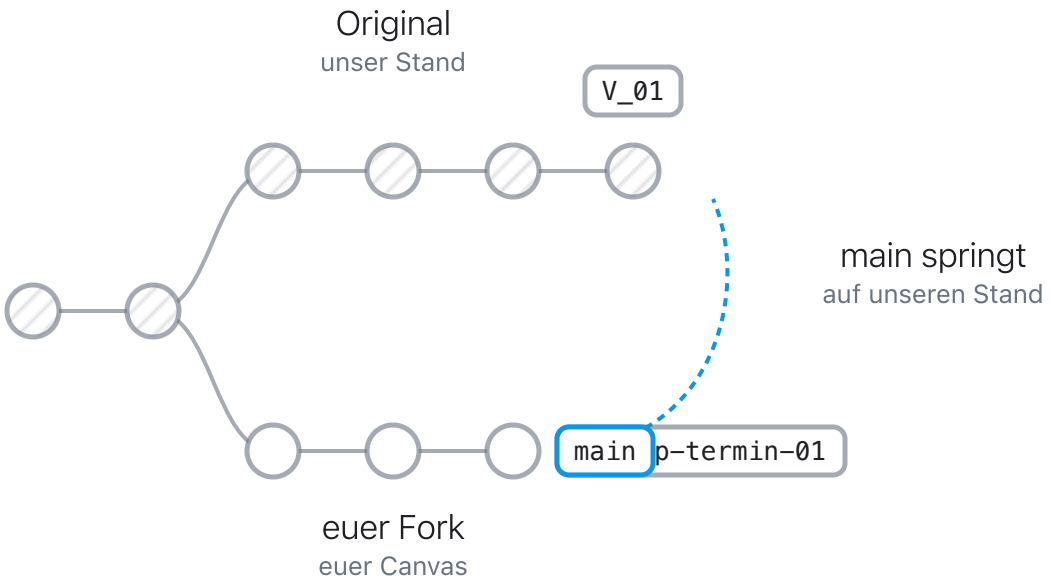
- Euer Canvas und unser Stand sind zwei verschiedene Projekte
- Fast jede Datei hätte einen Konflikt
- Wir wollen nicht zusammenführen, sondern **ersetzen**




Sync fork führt zusammen, was nicht zusammengehört.

# ÜBERSCHREIBEN: UNSEREN STAND ÜBERNEHMEN

1. **Euren Stand sichern** — Backup-Branch
2. **Klon auf unseren Stand setzen** — `reset --hard`
3. **Fork nachziehen** — `push --force`




**Merge**: beide Stände zusammenführen, beide Geschichten bleiben, Konflikte möglich. Gut im Team.

**Überschreiben**: euer Stand ist danach identisch mit dem Original, keine Konflikte. Gut für einen neuen Startpunkt.

# JETZT ALLE AUF DENSELBEN STAND

Im Terminal, im Projektordner — Zeile für Zeile oder alles auf einmal:

Terminal

```
git remote add upstream https://github.com/hfg-protolab/computationalThinking4.git
git add -A
git commit -m "Mein Stand Termin 1"
git branch backup-termin-01
git push origin backup-termin-01
git fetch upstream
git reset --hard upstream/main
git push --force origin main
npm i
```

→ Ab jetzt vor jeder Vorlesung nur noch: `npm run sync`

# BESCHREIBT EUER GUMMIBÄRCHEN




Zu viert, eine Packung:

- Wie würdet ihr die **Packung** beschreiben?
- Wie würdet ihr **ein** Gummibärchen beschreiben?
- Was ist bei **allen** Bärchen gleich, was ist **unterschiedlich**?

→ Nicht essen, bevor ihr es beschrieben habt.

# OBJEKT, ATTRIBUT, WERT

Ein Gummibärchen ist ein **Objekt**. Es hat **Attribute** — und jedes Attribut hat einen **Wert**.

| Attribut | Wert |
|---|---|
| Farbe | rot |
| Geschmack | Himbeere |
| Form | Bär |
| Zucker drauf | nein |

# KLASSE UND OBJEKT

Die Klasse beschreibt, welche Attribute ein Gummibärchen *hat*. Ein **Objekt** beschreibt, welche Werte *ein* Bärchen hat.

Klasse

```
GummyBear
  color:  text
  flavor: text
  sugar:  true / false
```

Object

```
GummyBear
  color:  red
  flavor: raspberry
  sugar:  true

GummyBear
  color:  green
  flavor: apple
  sugar:  false
```

# DIE PACKUNG ENTHÄLT BÄRCHEN

Auch die Packung ist ein Objekt — und eines ihrer Attribute sind **andere Objekte**.

Klasse

```
Bag
  brand:  text
  weight: number
  bears: GummyBear[]
```

# EINE BASIS, MEHRERE VARIANTEN

Saure Bärchen haben **alles**, was normale haben — **plus** eine Zuckerschicht.

Klasse

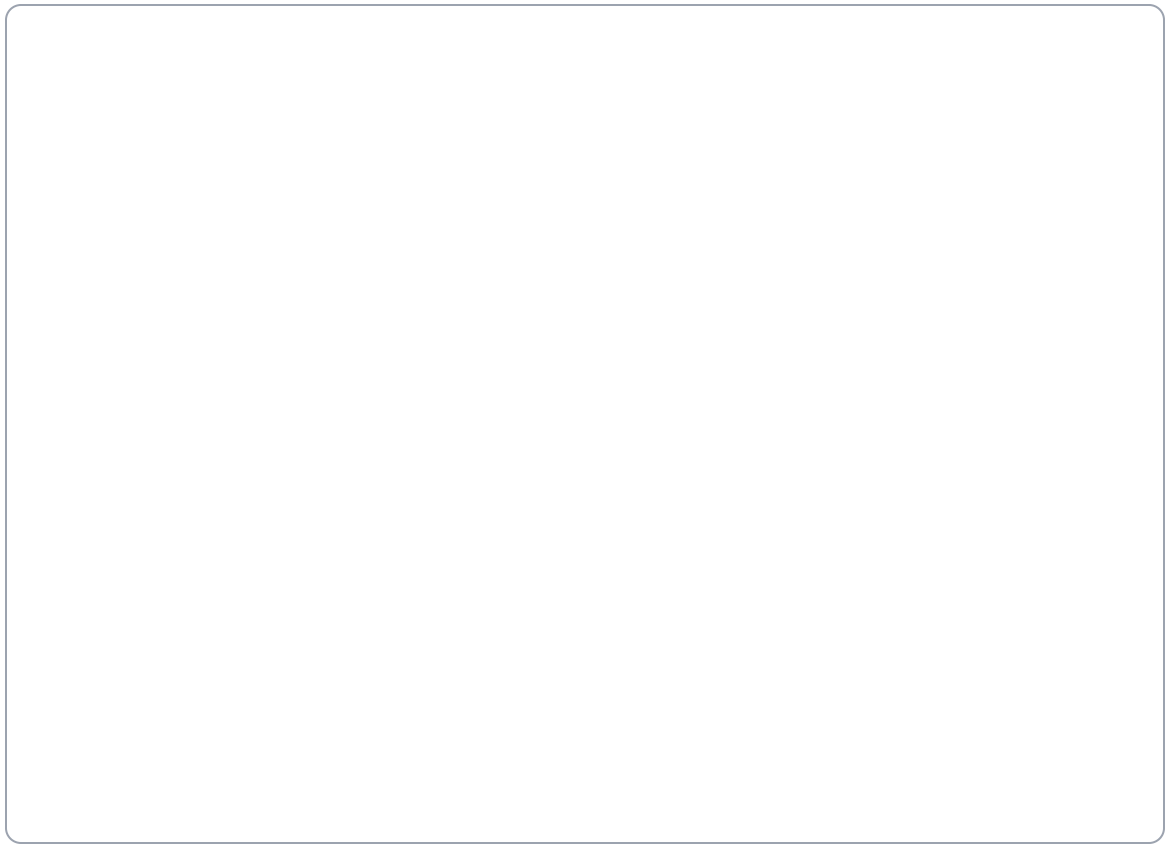
```
GummyBear
  shape: bear
  color:  text
  flavor: text

SourBear extends GummyBear
  sugarCoatingPercentage: number
```

# RÜCKBLICK

Letztes Mal: ein endloser Canvas mit einem Post-it — einfach drauflos gepromptet.

- Claude baut mehr oder etwas anderes als gewollt
- viel Hin und Her beim Korrigieren
- jede*r hatte am Ende eine andere Struktur im Code



Ausprobieren: ziehen, Doppelklick zum Bearbeiten, Ecken zum Skalieren

# AUF DEM BOARD

Jedes Post-it ist ein Objekt. Und es gibt mehr als Post-its.

Klasse

```
CanvasObject
  id:     text
  x, y, width, height: number

TextObject extends CanvasObject
  text:     text
  fontSize: number

StickyNote extends TextObject
  color: text

Circle extends CanvasObject
  color: text
```

Klasse

```
Board

  objects: CanvasObject []
```

# KOMPONENTEN SIND AUCH OBJEKTE

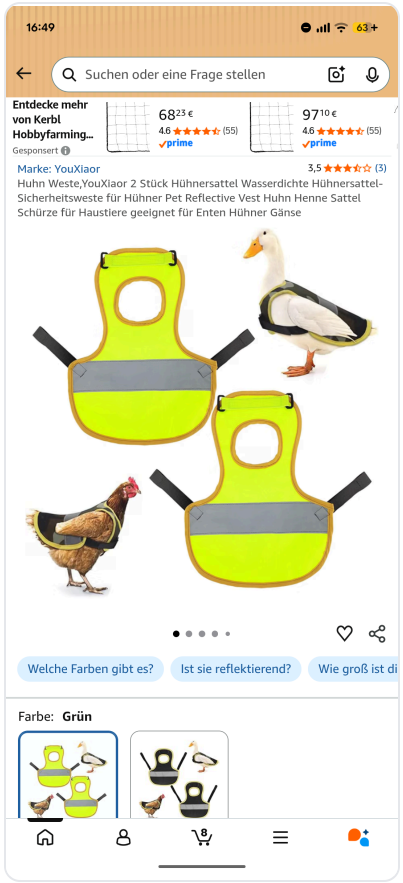
- **Basis-Komponente**: Position, Auswahlrahmen, Ziehen, Eckgriffe — für alle gleich
- **Varianten**: Post-it, Text, Kreis — bringen nur ihr Aussehen mit

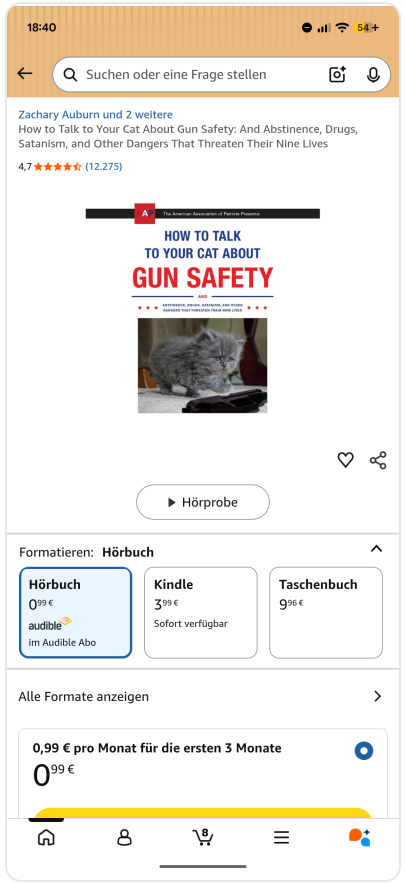
Pseudocode

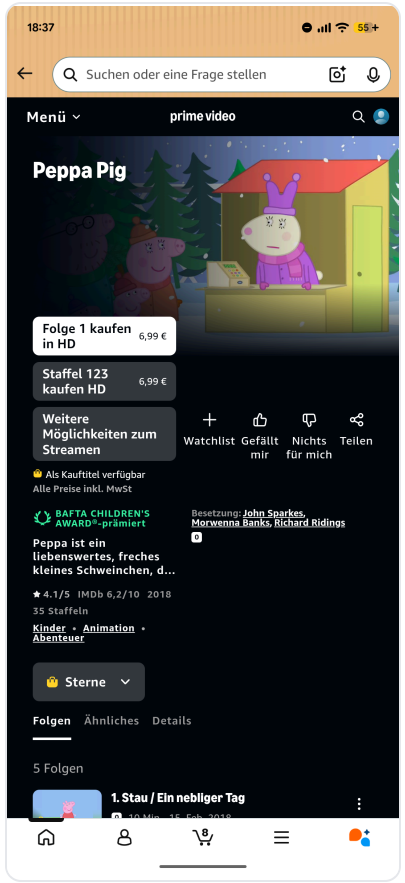
```
variants
  sticky-note → StickyNote
  text        → PlainText
  circle      → Circle

for each object in board: draw variants[type]
```

# ANDERES BEISPIEL



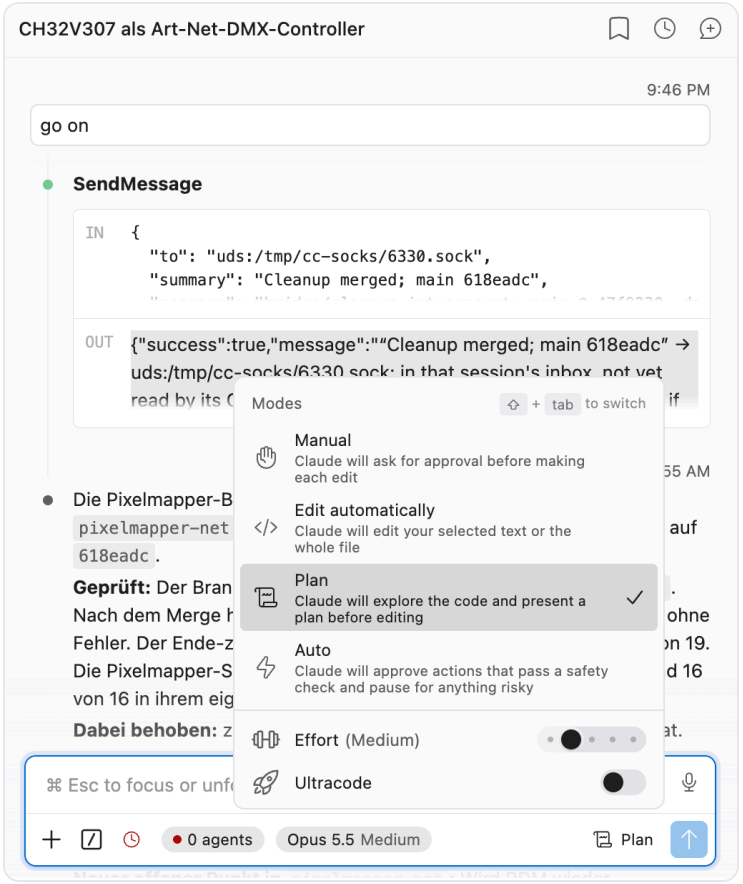

- **Basis** Product: Name, Preis, Bild, Bewertungen
- **Varianten**: Kleidung (+ Größe), Buch (+ ISBN, Seiten), Kindle-E-Book (+ Download, kein Versand)
- Warenkorb und Kasse brauchen nur die Basis: Name und Preis
- Eine neue Produktart braucht keinen neuen Warenkorb

# PLANNING MODE

Unten im Eingabefeld den Modus auf **Plan** stellen.

1. Claude liest sich ein und fragt nach, wo etwas unklar ist
2. Claude schreibt einen **Plan** — noch kein Code
3. Ihr lest den Plan und kommentiert, was nicht passt
4. Erst wenn der Plan stimmt: freigeben, Claude baut

→ Korrigieren ist im Plan billig, im Code teuer.

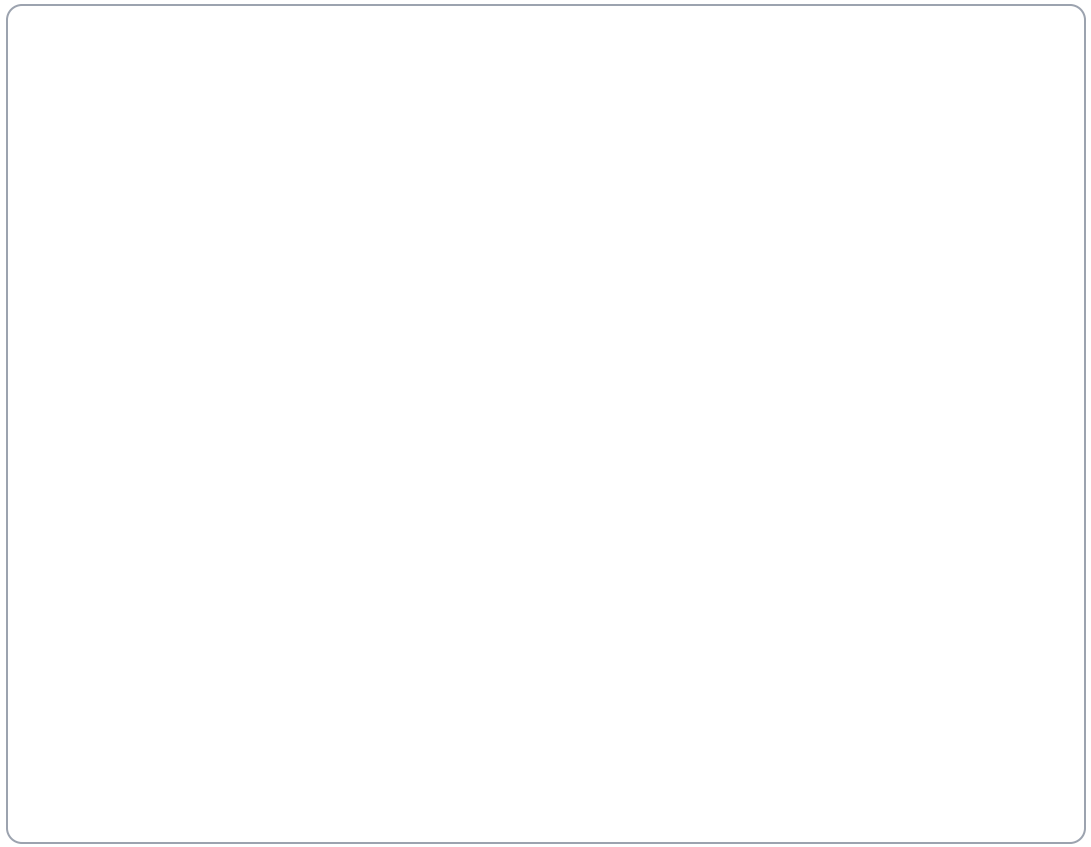


# AUFGABE: OBJEKTE AUF DEM BOARD

Im **Planning Mode**:

1. Eine Basis für alle Objekte — das Post-it wird eine Variante davon
2. Zwei weitere Varianten: **Text** und **Kreis** — und eine Leiste, um alle drei anzulegen
3. Das Board zeichnet alle über **eine** Liste von Varianten

Plan prüfen: Gibt es eine Basis? Hat der Kreis Text? Weiß das Board, was es zeichnet?



Ausprobieren: unten anlegen, ziehen, an den Ecken skalieren, Doppelklick zum Bearbeiten

# STATE



Scheiße kein Klopapier mehr

# STATE



→ Im Bad stapelt sich das Klopapier.

**Wie lösen wir das Problem, dass wir nicht zu viel Klopapier kaufen?**

# STATEFUL UND STATELESS

**Stateful** — die Liste im Kopf

Jede*r merkt sich **selbst**, was fehlt.

Zwei Köpfe, zwei Listen → viel Klopapier.

**Stateless** — die geteilte Liste

Niemand merkt sich selbst etwas. Was fehlt, steht an **einer** Stelle.

Wer etwas kauft, hakt es dort ab → **alle** sehen es.

Und auf dem Canvas: Merkt sich jedes Post-it selbst, wo es liegt — oder gibt es eine Liste für alle?

# EIN STATE FÜR DAS GANZE BOARD

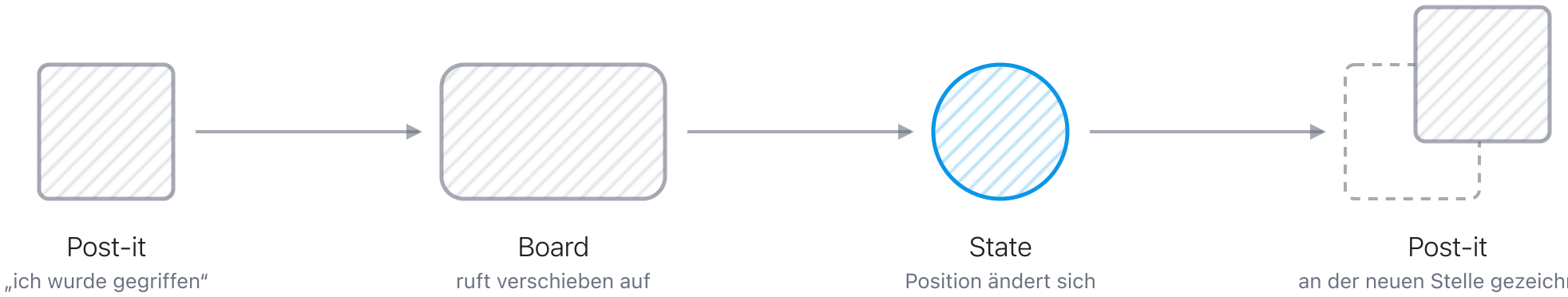
- **Eine** Stelle weiß, welche Objekte es gibt und wie sie aussehen
- Die Komponenten bekommen nur, was sie anzeigen sollen
- Ändern geht **nur** über feste Funktionen

Pseudocode

```
Board
  objects: Liste von CanvasObject

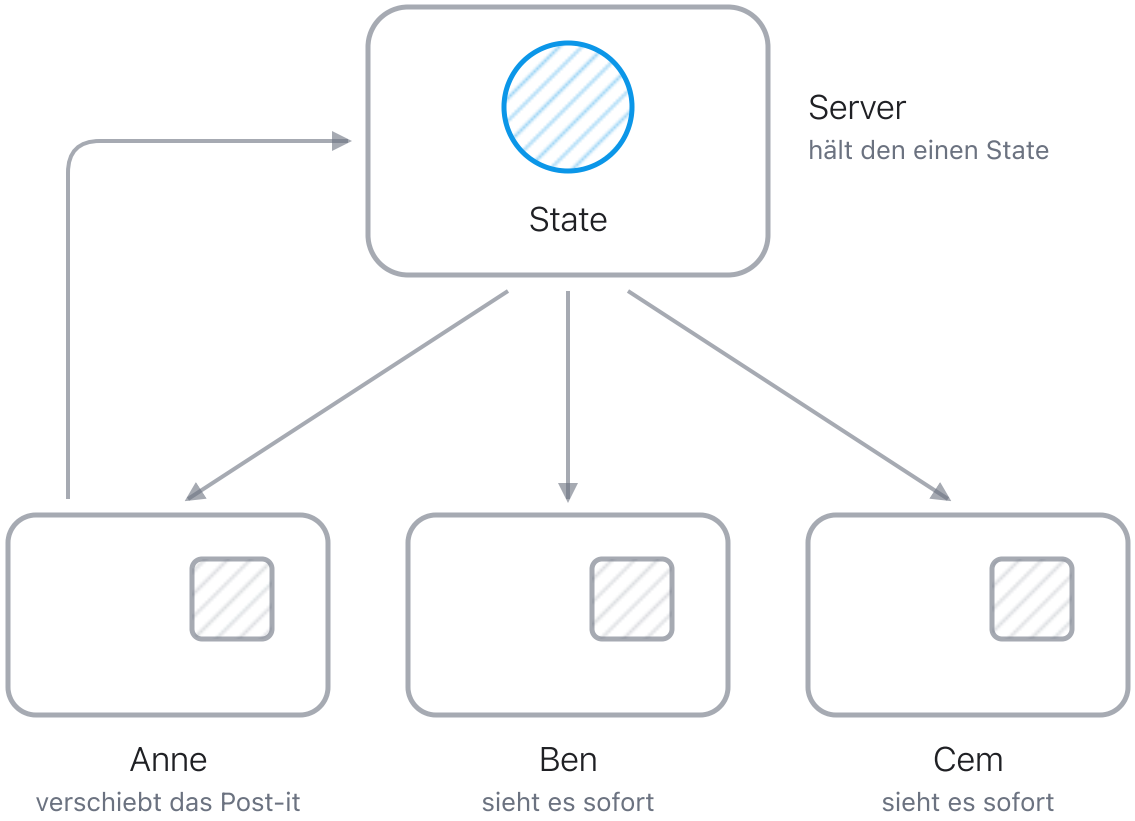
ändern nur über
  addObject(objekt)
  moveObject(id, x, y)
  updateText(id, text)
```

# WAS BEIM VERSCHIEBEN PASSIERT




→ Das Post-it bewegt sich nicht selbst. Es bewegt sich, **weil sich der State ändert**.

# WARUM DAS FÜR ZUSAMMENARBEIT WICHTIG IST




- Der State liegt später **auf dem Server** und wird an alle verteilt
- Eigene Daten in Komponenten → jede*r sieht etwas anderes
- Reine Daten: **speichern, verschicken, rückgängig machen**

# WAS GEHÖRT IN DEN GETEILTEN STATE?

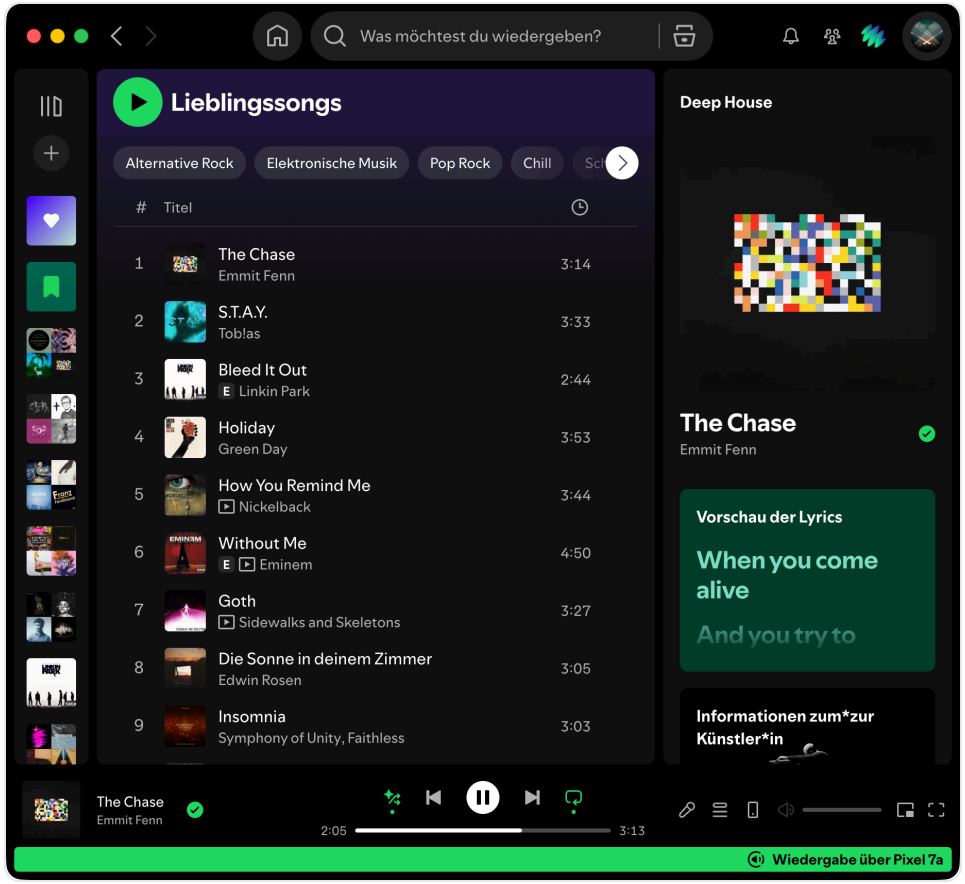
Wenn zwei Leute am selben Board arbeiten — sehen beide …

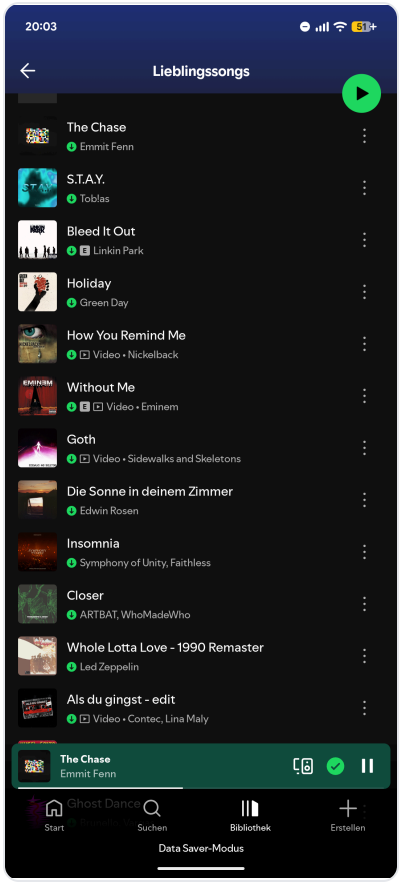
# WAS GEHÖRT IN DEN GETEILTEN STATE?

Wenn zwei Leute am selben Board arbeiten — sehen beide …

- welche Post-its es gibt?
- welches Post-it ich gerade **ausgewählt** habe?
- wie weit ich **reingezoomt** habe?
- an welcher Stelle des Boards ich gerade **schaue**?
- dass ich gerade einen Text **bearbeite**?

# ANDERES BEISPIEL




## Ein State, alle Geräte

- Gleicher Song, gleiche Stelle im Lied
- Pause am Handy → Pause am Laptop
- Umschalten aufs andere Gerät

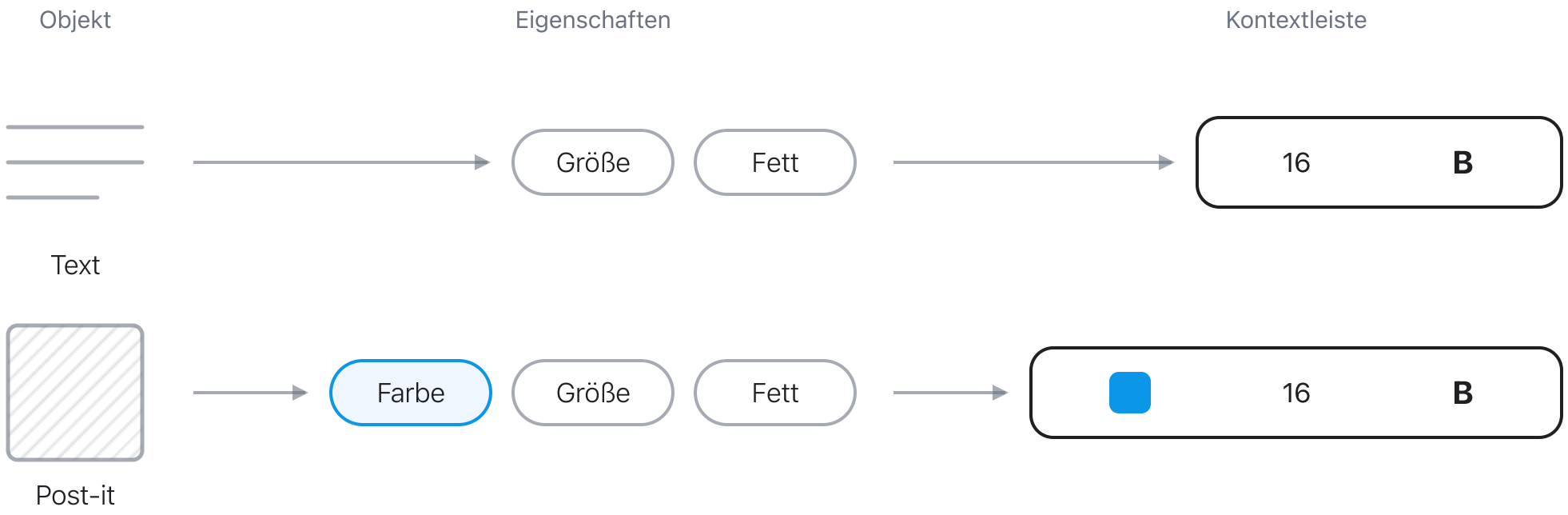
Fast alle modernen Apps bauen so

# AUFGABE: EIN STATE FÜR DAS BOARD

Im **Planning Mode**:

1. Ein zentraler State/ Zustand mit allen Objekten — ändern nur über Funktionen
2. Die Objekte werden **stateless**: anzeigen und Events melden
3. Auswahl, Zoom und Ansicht bleiben **lokal**

# UI AUS DEM OBJEKT



# DIE KONTEXTLEISTE

Wählt man ein Objekt aus, zeigt die Leiste nur, was zu ihm passt.

| | Farbe | Größe | Fett |
|---|---|---|---|
| Post-it | ✓ | ✓ | ✓ |
| Text | – | ✓ | ✓ |
| Kreis | ✓ | – | – |

Nicht „bist du ein Post-it?“, sondern „**hast du** eine Farbe?“

Pseudocode

```
editors
  color    → ColorEditor
  fontSize → FontSizeEditor
  bold     → BoldEditor

zeige jeden Editor,
dessen Attribut das Objekt hat
```

# TASTENKÜRZEL: AUCH EINE LISTE

Wie bei der Kontextleiste: **eine Liste** — jede Zeile verbindet eine Taste mit einer Aktion.

- Ein neuer Shortcut = **eine neue Zeile**
- Die Aktionen brauchen nur die Basis: `id`, `x`, `y` → sie gelten für **jede** Variante
- Sie laufen über den State: ← ruft dasselbe `moveObject` auf wie das Ziehen

Pseudocode

```
shortcuts
  mod+C        → copy
  mod+X        → cut
  mod+V        → paste
  mod+D        → duplicate
  Delete       → remove
  Escape       → deselect
  ← ↑ → ↓      → move one grid step

on key: run the action
        on the selected object
```

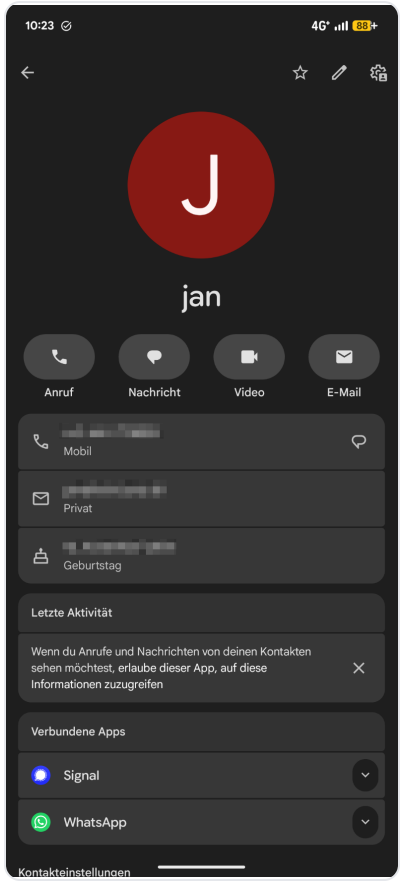
# ERWEITERN, OHNE ZU ÄNDERN

Eine neue Eigenschaft **Border** — in drei Schritten:

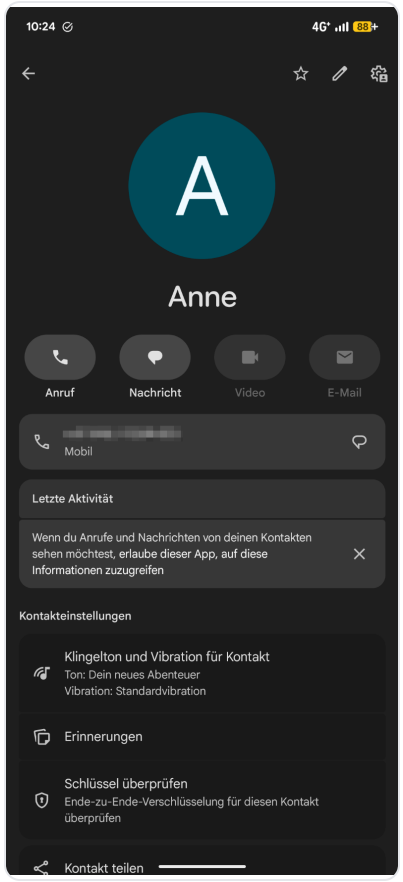
1. `border` in den Klassen der Objekte, die einen Rand haben sollen
2. Einen Editor für den Rand bauen
3. Eine Zeile in die Liste: `border → BorderEditor`

→ Die Leiste selbst und die anderen Objekte bleiben **unverändert**.

# DASSELBE IN DER KONTAKTE-APP




Jan: Nummer, Mail, Apps




Anne: nur Nummer

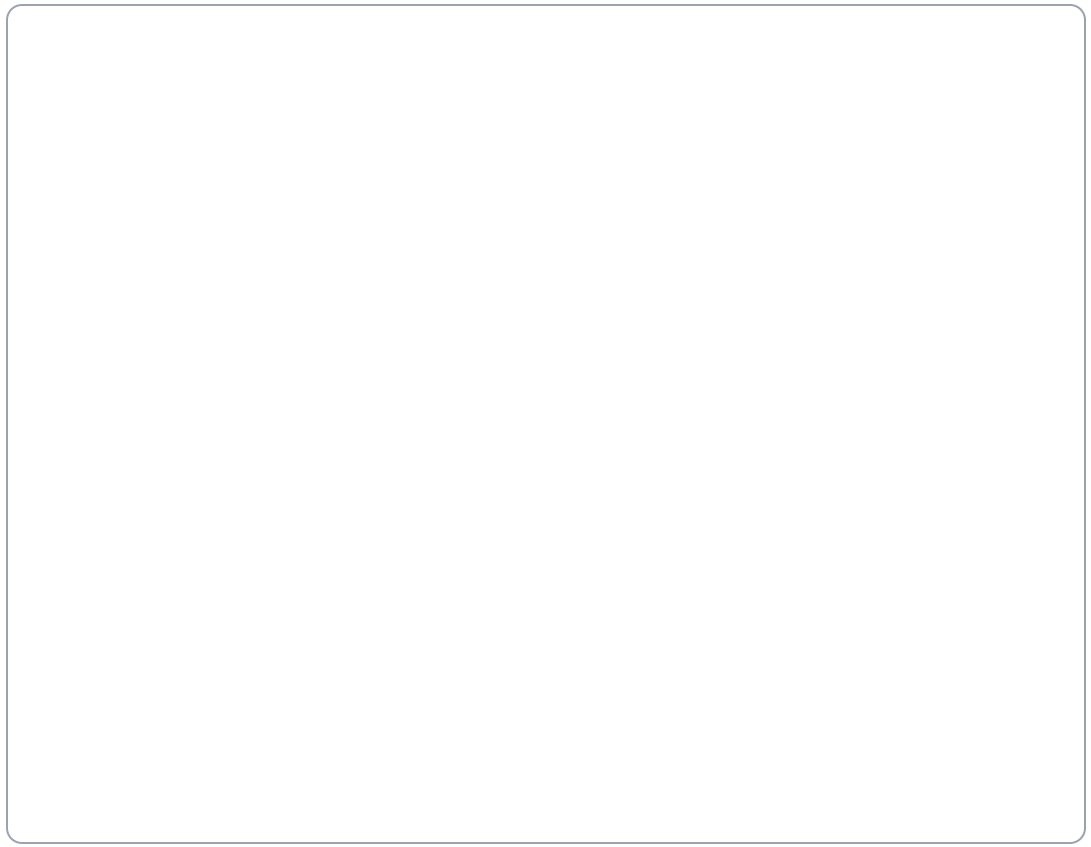
- Nummer gespeichert → **Anruf**, **Nachricht**
- E-Mail gespeichert → **E-Mail**
- Signal, WhatsApp → **Verbundene Apps**

→ Die Seite zeigt nur, was für den Kontakt **hinterlegt ist**.

# AUFGABE: DIE KONTEXTLEISTE

Im **Planning Mode**:

1. Eine Leiste über dem gewählten Objekt
2. Sie zeigt Editoren für **Farbe**, **Schriftgröße, Border** und **Fett** — aber nur, wenn das Objekt das Attribut hat
3. Kreise mit Border, Post-its ohne
4. Welcher Editor zu welchem Attribut gehört, steht in **einer Liste**
5. **Shortcuts** als Liste: ⌘C, ⌘X, ⌘V, ⌘D, ⌫, Esc, Pfeiltasten



Ausprobieren: auswählen, Farbe, Größe oder Fett ändern — und Pfeiltasten, ⌘D, ⌫

# WAS WIR HEUTE GEBAUT HABEN

| Idee | Kernaussage |
| --- | --- |
| Objekte | Eine Basis, mehrere Varianten — alle mit derselben Schnittstelle |
| State | Ein zentraler State, die Komponenten zeigen ihn nur an |
| UI aus dem Objekt | Listen statt if-Ketten — erweitern, ohne zu ändern |