Domain Driven Design of Large Software Systems

Master Digital Sciences – Software Architecture, WS 22/23

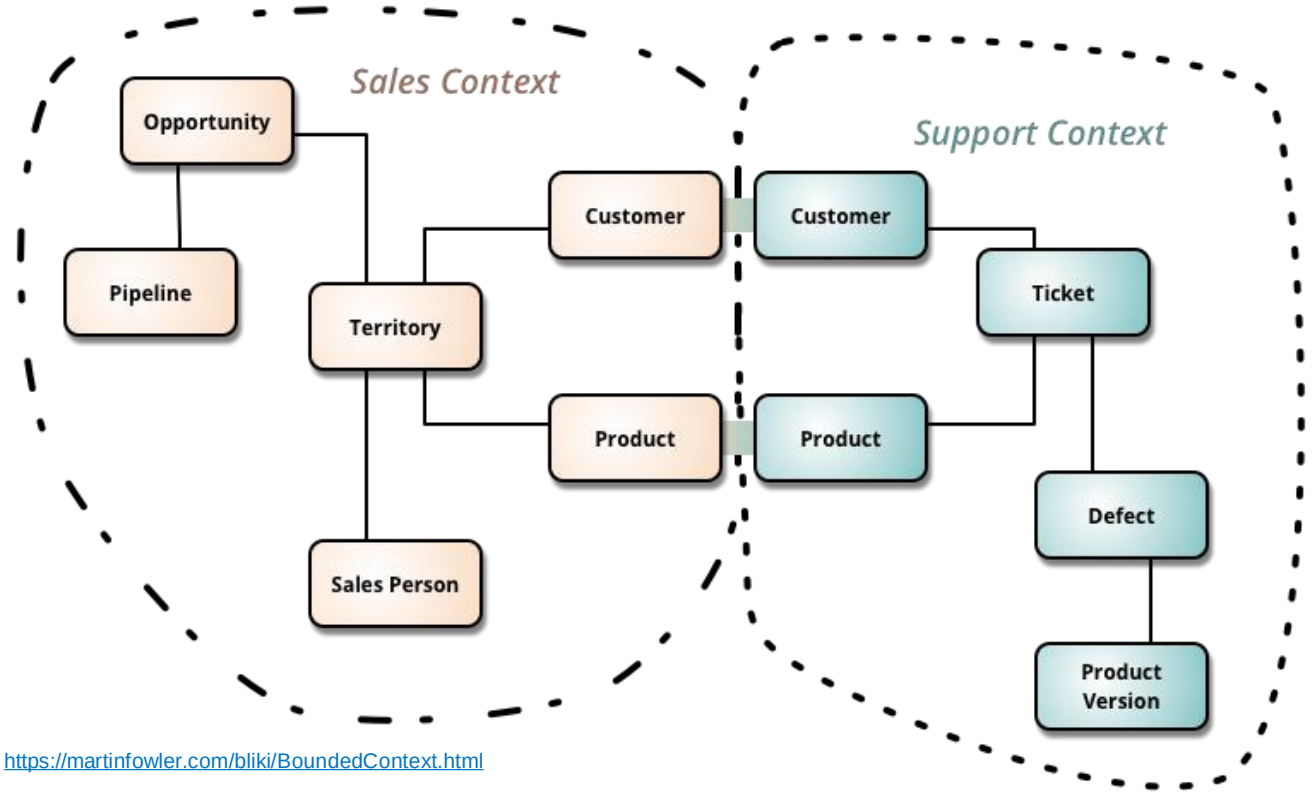
Prof. Dr.-Ing. Stefan Bente

Context Map

Technology
Arts Sciences
TH Köln

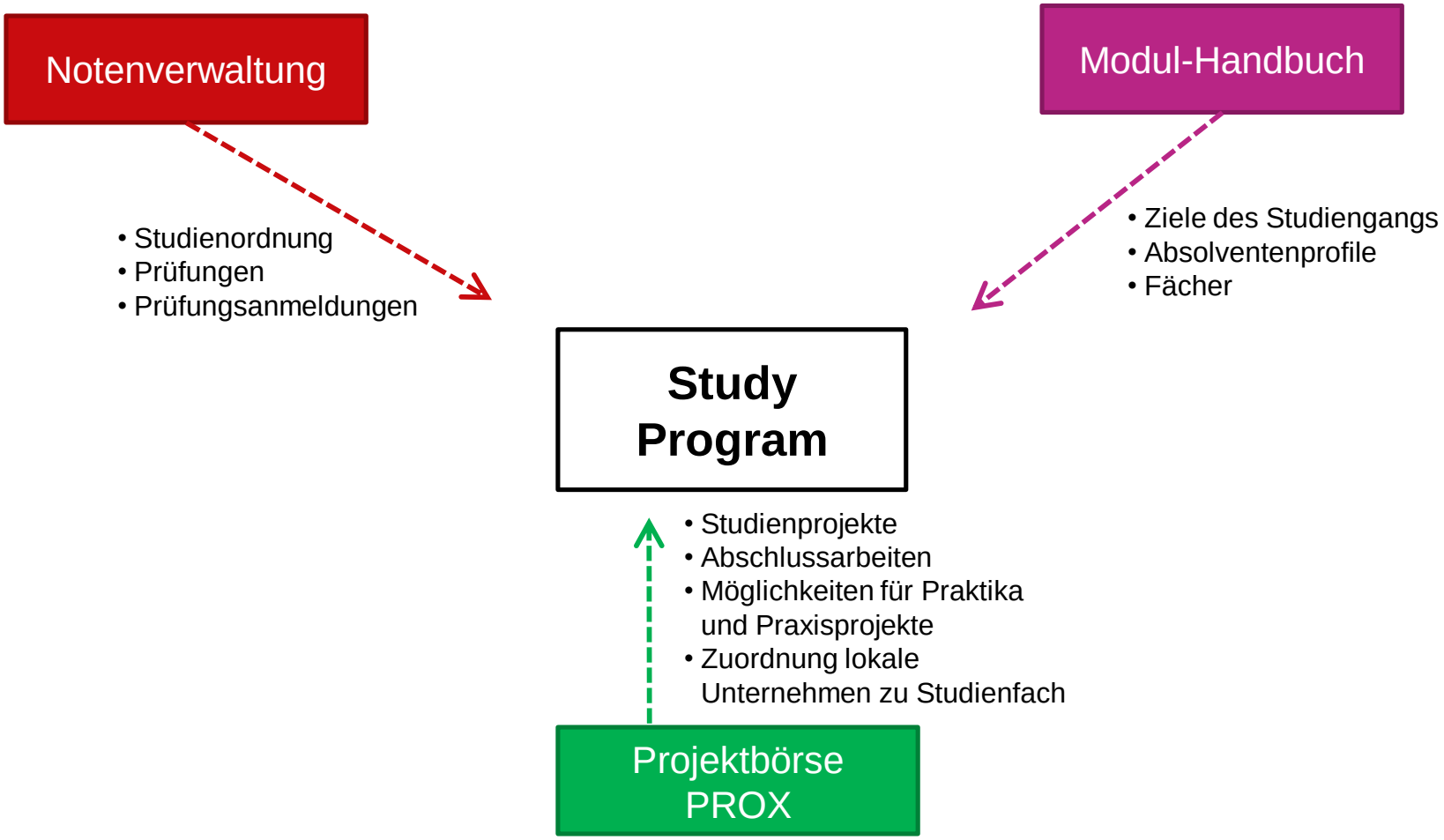
Definition: **Context Map**

1. DDD's strategic design goes on to describe a variety of ways that you have relationships between Bounded Contexts. It's usually worthwhile to depict these using a context map.
2. The Context Map […] is […] a simple diagram that shows the mappings between two or more existing Bounded Contexts.

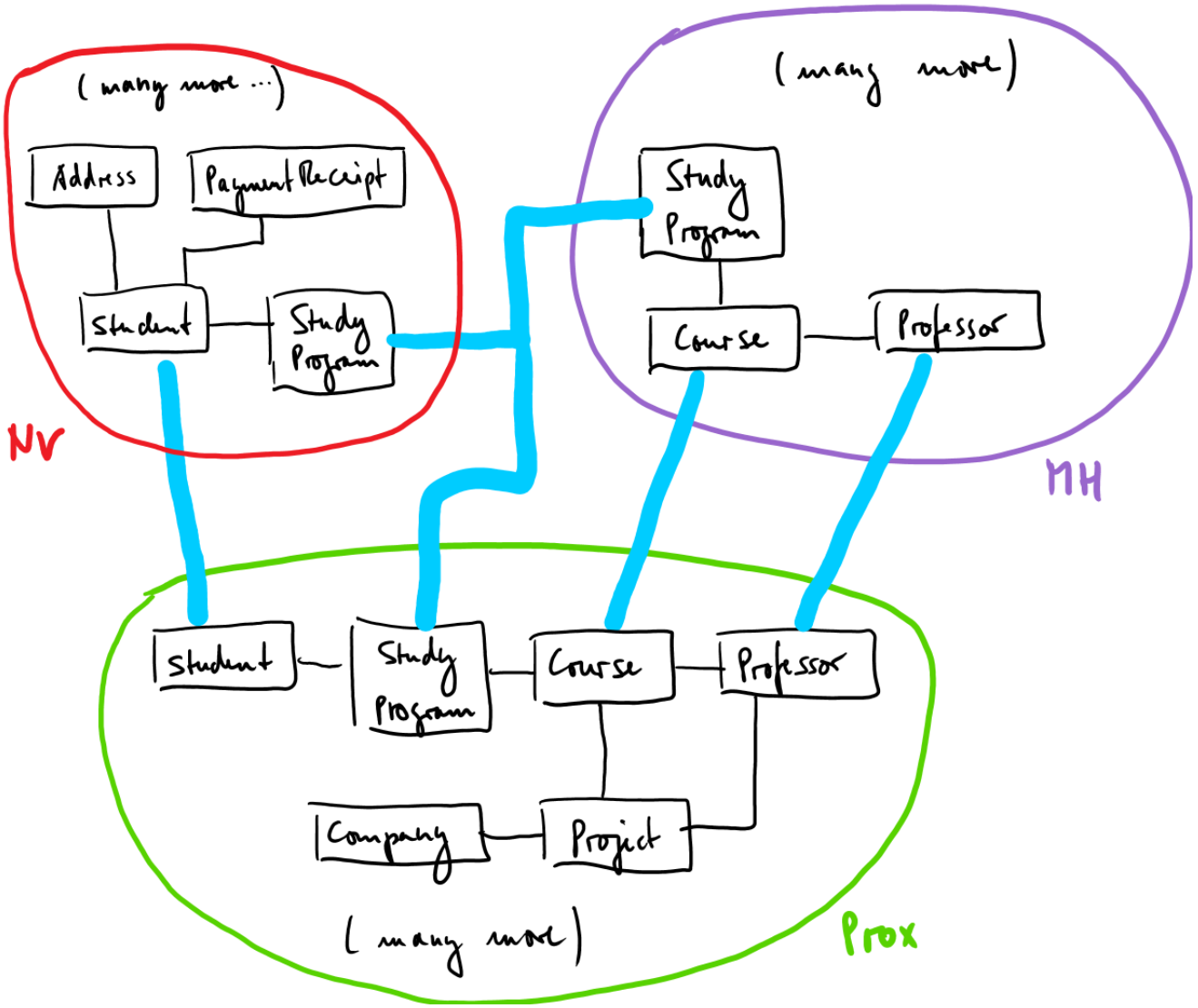


Quellen:
1.und Bild: Martin Fowler, https://martinfowler.com/bliki/BoundedContext.html
2.[Vernon], p. 87

Für Einfachheit: **Wir betrachten nur 3 Systeme weiter …**



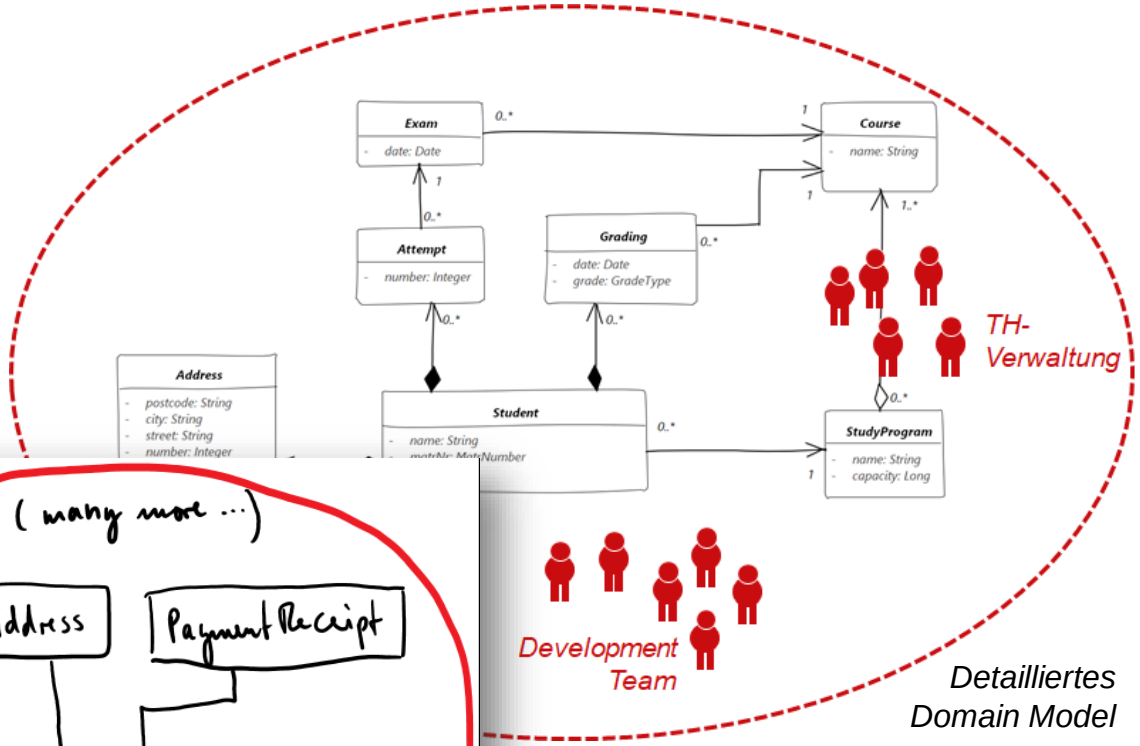
Context Map für die Hochschulsysteme

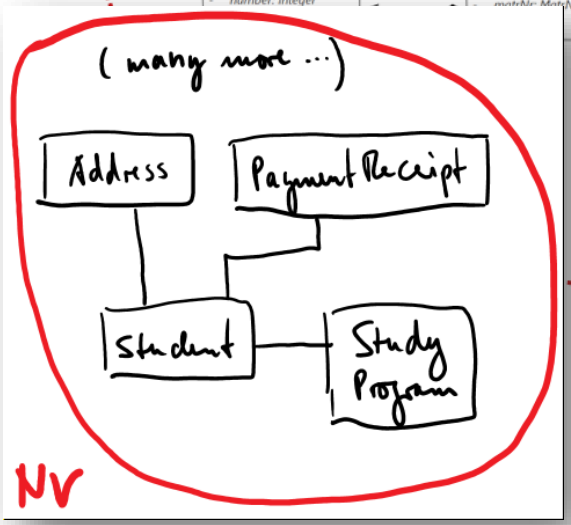


- Die Context Map zeigt die Beziehung der einzelnen Bounded Contexts zueinander.
- Das "Innenleben" der Bounded Contexts wird nur stark vereinfacht gezeigt. Es ist gerade **nicht** der Anspruch, ein konsistentes Gesamtmodell zu spezifizieren.
- **Innerhalb** eines Bounded Context ist das Modell u.U. viel detaillierter (siehe nachfolgende Folie)

Context Map dient zur Diskussion der Integration

- *Zum Vergleich:* Context Map – detailliertes Domain Model
- In der Context Map geht es um die Integration zwischen den Bounded Contexts



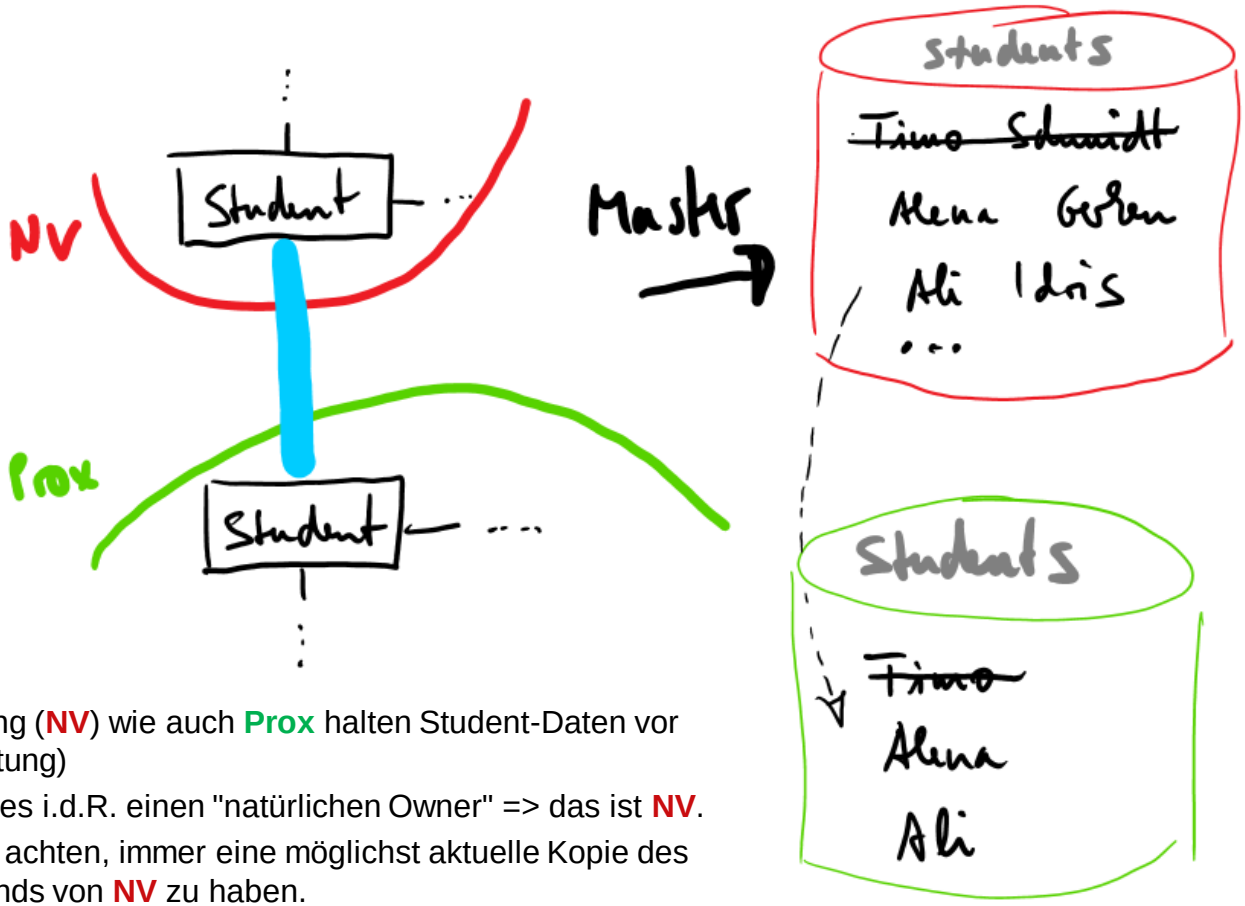


Auszug aus Context Map

Technology Arts Sciences TH Köln

Integrations-Pattern zwischen Bounded Contexts

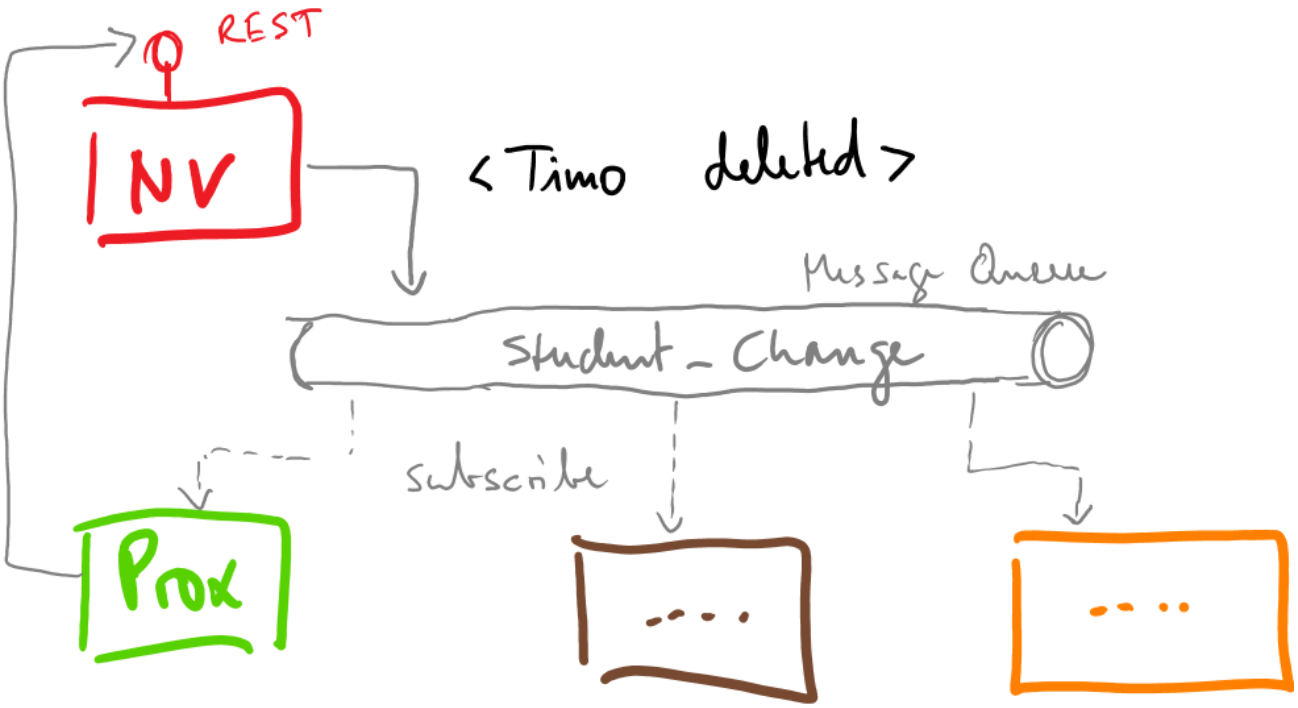
Was heißt das konkret für "Student" bei NV / Prox?



- Sowohl Notenverwaltung (**NV**) wie auch **Prox** halten Student-Daten vor (Redundante Datenhaltung)
- Im Gesamtsystem gibt es i.d.R. einen "natürlichen Owner" => das ist **NV**.
- D.h. **Prox** muss darauf achten, immer eine möglichst aktuelle Kopie des Studenten-Datenbestands von **NV** zu haben.

1. Beim initialen Start der Services holt sich Prox über einen REST-Call einen Abzug der Datenbasis (oder den für ihn relevanten Teil der Daten …)
2. Bei der Änderung eines Student in **NV** (z.B. Timo exmatrikuliert sich) schickt **NV** einen Event, z.B. "DELETE Timo". **Prox** (und ggfs. andere Systeme) subscriben diesen Event, haben einen Listener implementiert und reagieren darauf. Z.B. bei <Timo deleted> nimmt auch **Prox** Timo aus seinem Datenbestand heraus.

Vorgriff auf technische Realisierung: Synchron / asynchron



- Als Vorgriff auf die technische Realisierung von Bounded Contexts als möglichst unabhängige Systeme:

1. **NV** implementiert ein REST-API, um synchrone Queries und Abfragen zu ermöglichen. Diese Calls sollten aber möglichst wenig genutzt werden, da jeder synchrone Call auch immer eine enge Kopplung beinhaltet.
2. Als zweites Interface (neben dem synchronen REST) definiert **NV** eine Anzahl von fachlichen Events, wie z.B. die Exmatrikulation eines Students ("<student> deleted"). Bei <Timo deleted> nimmt auch **Prox** Timo aus seinem Datenbestand heraus.

Integration Patterns

maximaler Abstimmungsbedarf

- **Gemeinsamer Bounded Context**
  - Komplett gemeinsames Modell
- **Shared Kernel**
  - Teilmodell wird von zwei BC gemeinsam gepflegt
- **Customer / Supplier**
  - Kunde-Lieferant-Beziehung: Kunde bestimmt Struktur des übernommenen Teilmodells mit
- **Conformist**
  - Ein BC übernimmt das Teilmodell eines anderen BC
- **Seperate Ways**
  - Bounded Contexts sind vollständig entkoppelt

oft kombiniert mit

Anticorruption Layer

- Isolationschicht, um das lokale Modell gegen das externe Modell "abzuschirmen"

Open Host Service

- Satz von Services für den Zugriff von außen

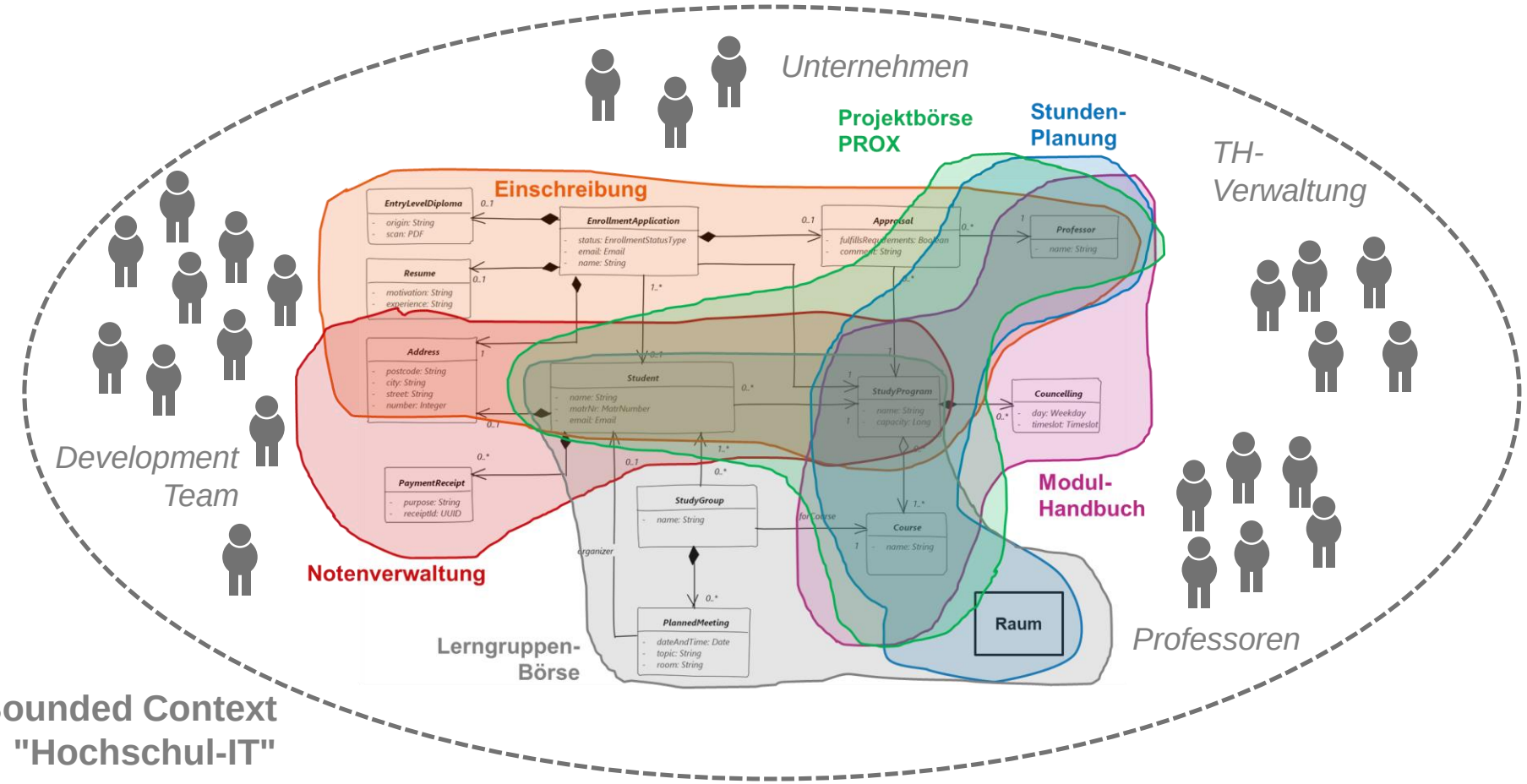
Published Language

- Gut dokumentierte Inhaltsformate ("gemeinsame Sprache" für den Austausch

kein Abstimmungsbedarf

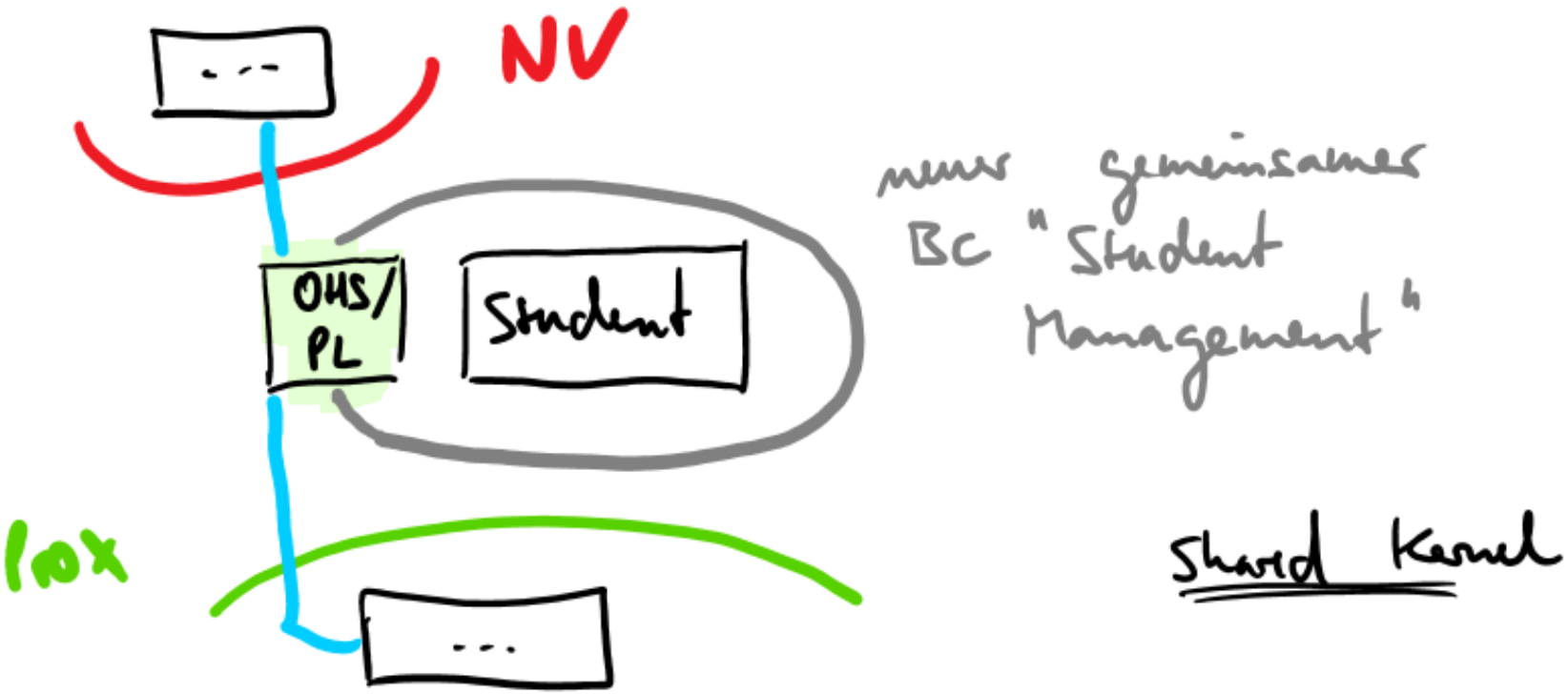
Quelle: angelehnt an [Evans], S. 388

Gemeinsamer Bounded Context



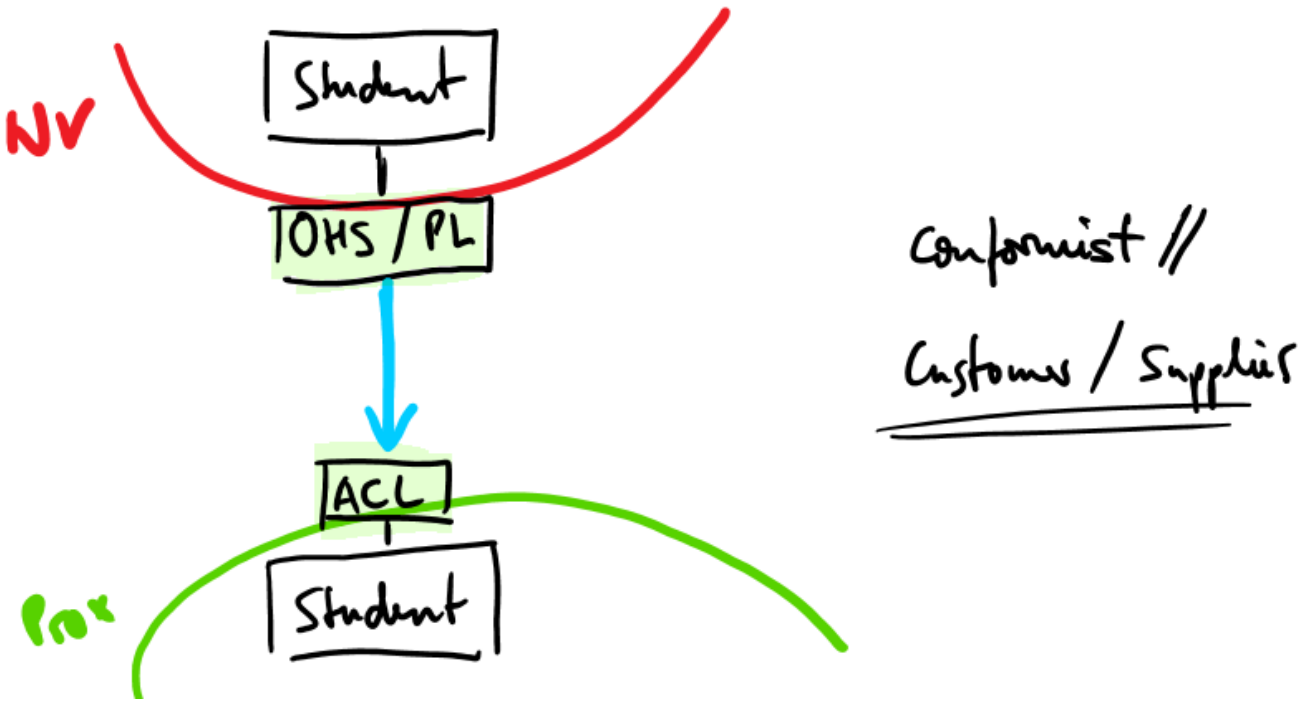
- Keine Unterteilung in lokale, abgegrenzte Kontexte (Sub-Domänen)
- Nur bei sehr kleinen Landschaften eine Option (hier eher nicht …)

Shared Kernel



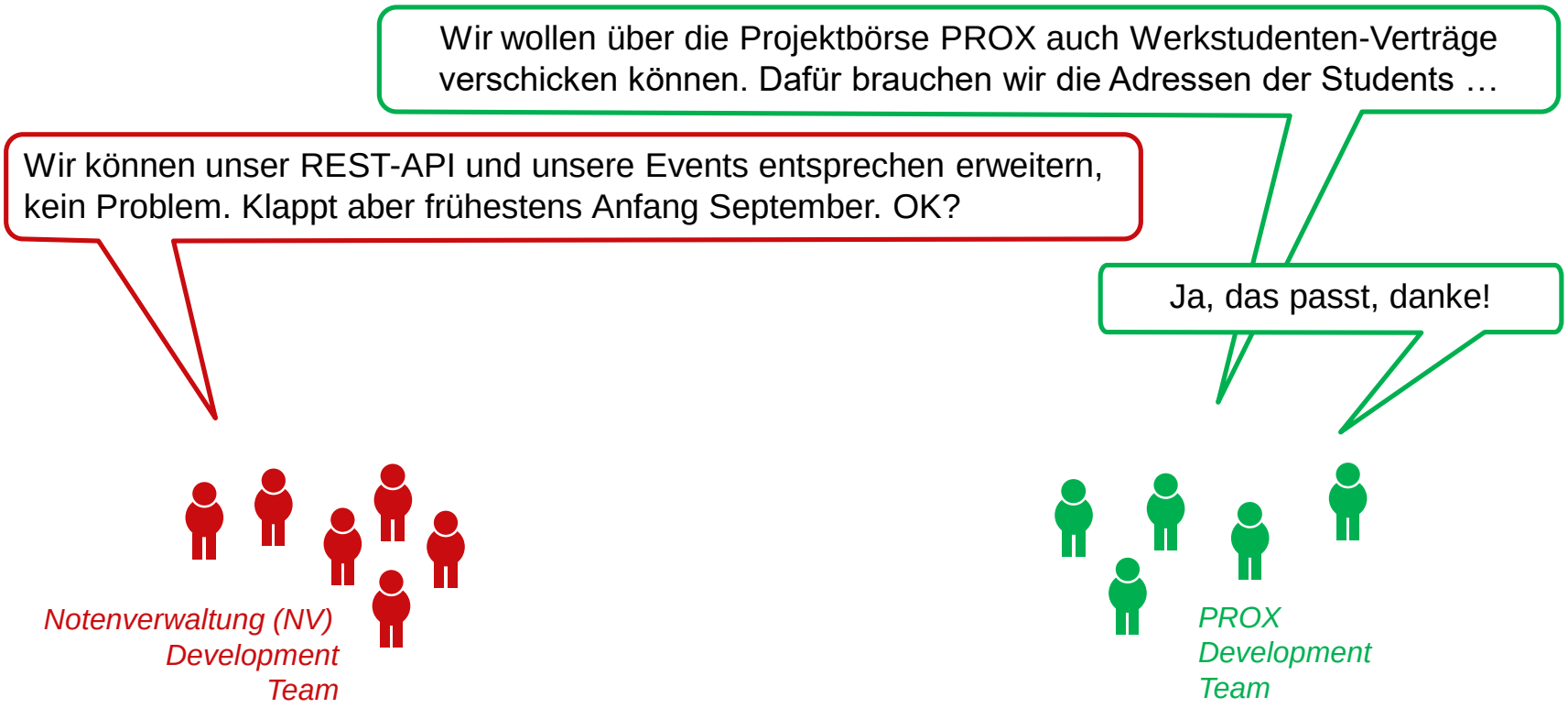
- **NV** und **Prox** einigen sich darauf, aus dem **`Student`**-Aggregate einen eigenen Bounded Context zu machen, der von beiden Development Teams gemeinsam bearbeitet wird.
- Faktisch wird damit der Grundstein für ein neues IT-System, dessen alleiniger Zweck die Verwaltung von Studierenden-Stammdaten ist. Es sollte über einen Open Host Service / Published Language ein Service-Interface nach außen zur Verfügung stellen.
- So ein Vorgehen kann in Einzelfällen sinnvoll sein. Im vorliegenden Fall wohl eher nicht, weil die Studenten-Stammdaten ein zu "dünnes" System ergeben.

Conformist / Customer-Supplier



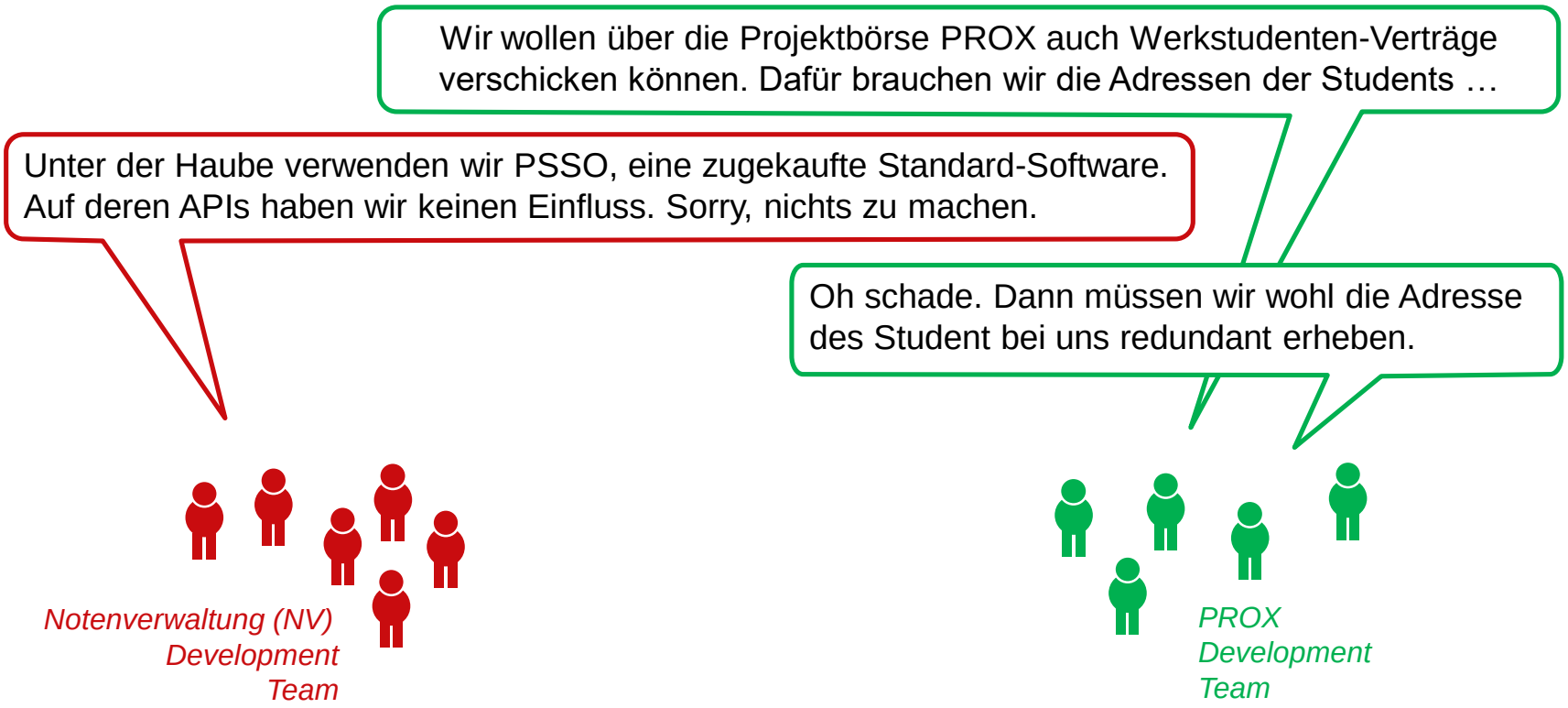
- **NV** und **Prox** haben jeweils ein eigenes **`Student`**-Aggregate, mit unterschiedlichen Attributen. Einer von beiden ist "upstream", der andere "downstream". Die "Upstream-Seite" ist die führende Seite – nur hier sollten Erzeugung und Löschung vorgenommen werden.
- In unserer Skizze ist upstream / downstream als Pfeil gekennzeichnet. **NV** ist upstream, weil hier Students angelegt werden. **Prox** repliziert sich von dort die Student-Daten.
- Auf **NV**-Seite wird über einen Open Host Service / Published Language ein Service-Interface nach außen zur Verfügung gestellt. **Prox** zieht einen Anticorruption-Layer ein, um sein "eigenes" Student-Modell behalten zu können, und nicht komplett dem NV-Modell folgen zu müssen.
- Conformist / Customer-Supplier haben die hier skizzierte Upstream/Downstream-Beziehung, unterscheiden sich aber in der Art, wie Änderungen ausgehandelt werden (siehe nächste Folien).

Customer-Supplier-Beziehung



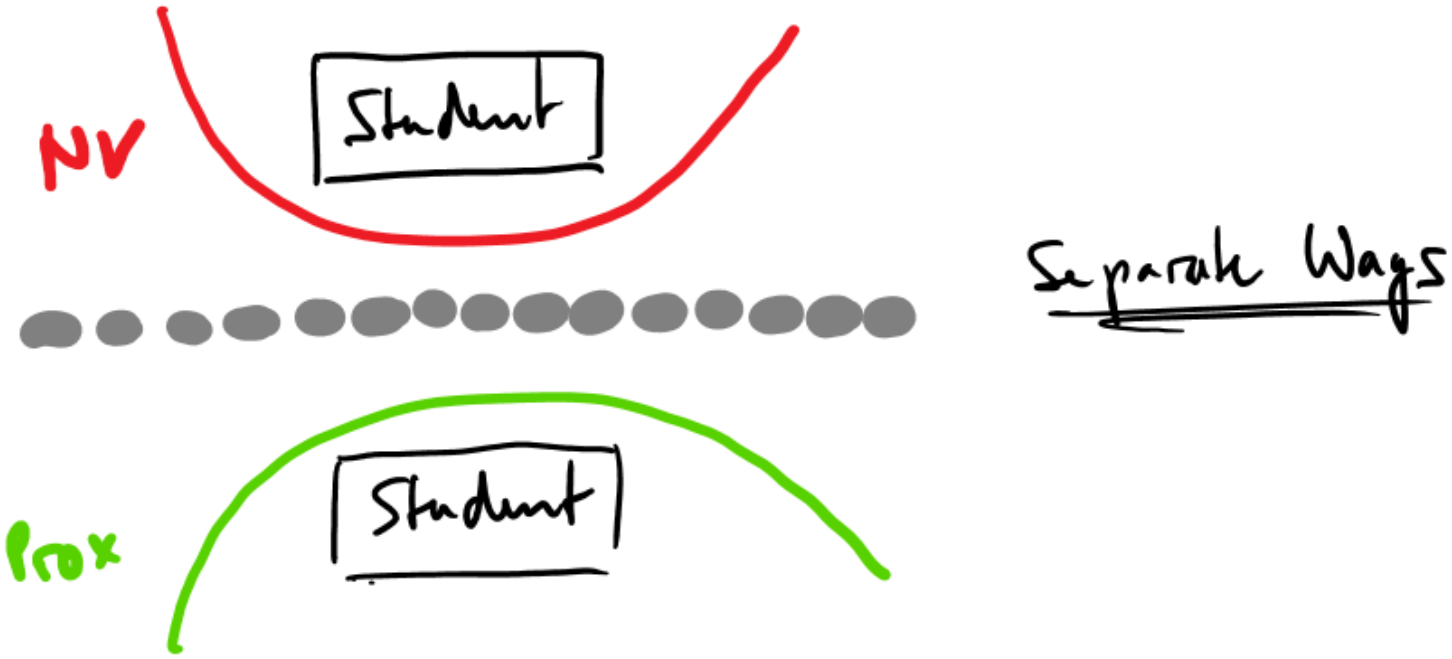
- Ein Bounded Context ist Lieferant (**Supplier**). Hier: **NV**.
- Der andere BC greift als Kunde (**Customer**) darauf zu. Hier: **PROX**
  - Lieferant hat Ownership über die Datenmodellierung
  - Kunde gibt Anforderungen über Änderungen und Erweiterungen
- *Empfehlung*: verwenden, wenn
  - Ownership geklärt, und Team des Lieferanten zu Änderungen bereit (z.B. in gleicher Firma)

Conformist-Beziehung



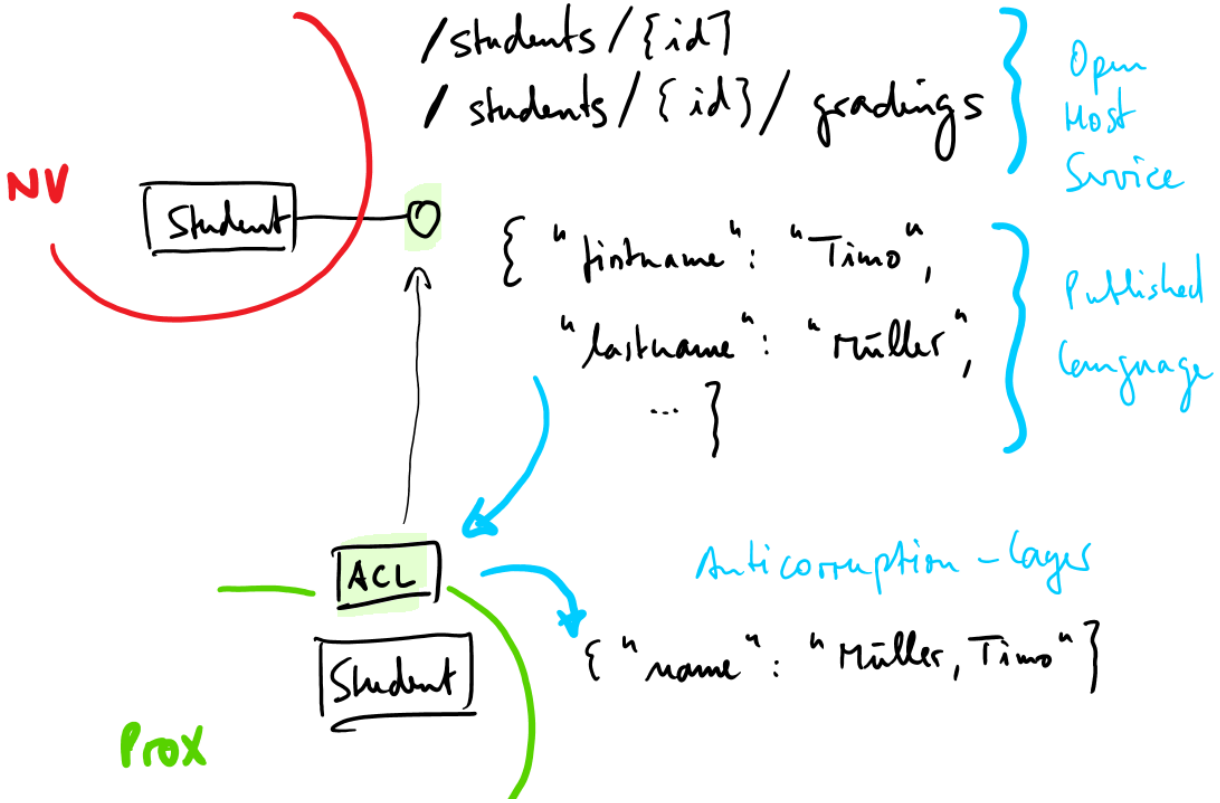
- Konsument (downstream) akzeptiert vorbehaltlos (Teil-)Datenmodell des Providers
- Keine Kundenbeziehung mit Änderungswünschen auf Kundenseite
- *Empfehlung:* verwenden, wenn Lieferant Änderungswünsche nicht akzeptiert
  - z.B. bei externen, unabhängigen Providern

Separate Ways



- **NV** und **Prox** haben jeweils ein eigenes **`Student`**-Aggregate, das sie **nicht miteinander abgleichen**.
- Beide Modelle und beide Datenhaltungen existieren vollständig unabhängig von einander.
- Das ist die Option, die am meisten Freiheit und am wenigsten Abstimmungsbedarf kostet. Andererseits ist es unter Umständen nicht die Funktionalität, die man sich wünscht.
- **IN DER ECHTEN WELT** … gibt es Notenverwaltung (= PSSO) und PROX (entwickelt bei ArchiLab / Labor Prof. Bente) wirklich, und sie haben ein "Separate Ways"-Verhältnis, was Student angeht (und alles andere auch). Alles andere wäre in der Hochschul-Realität gar nicht machbar.

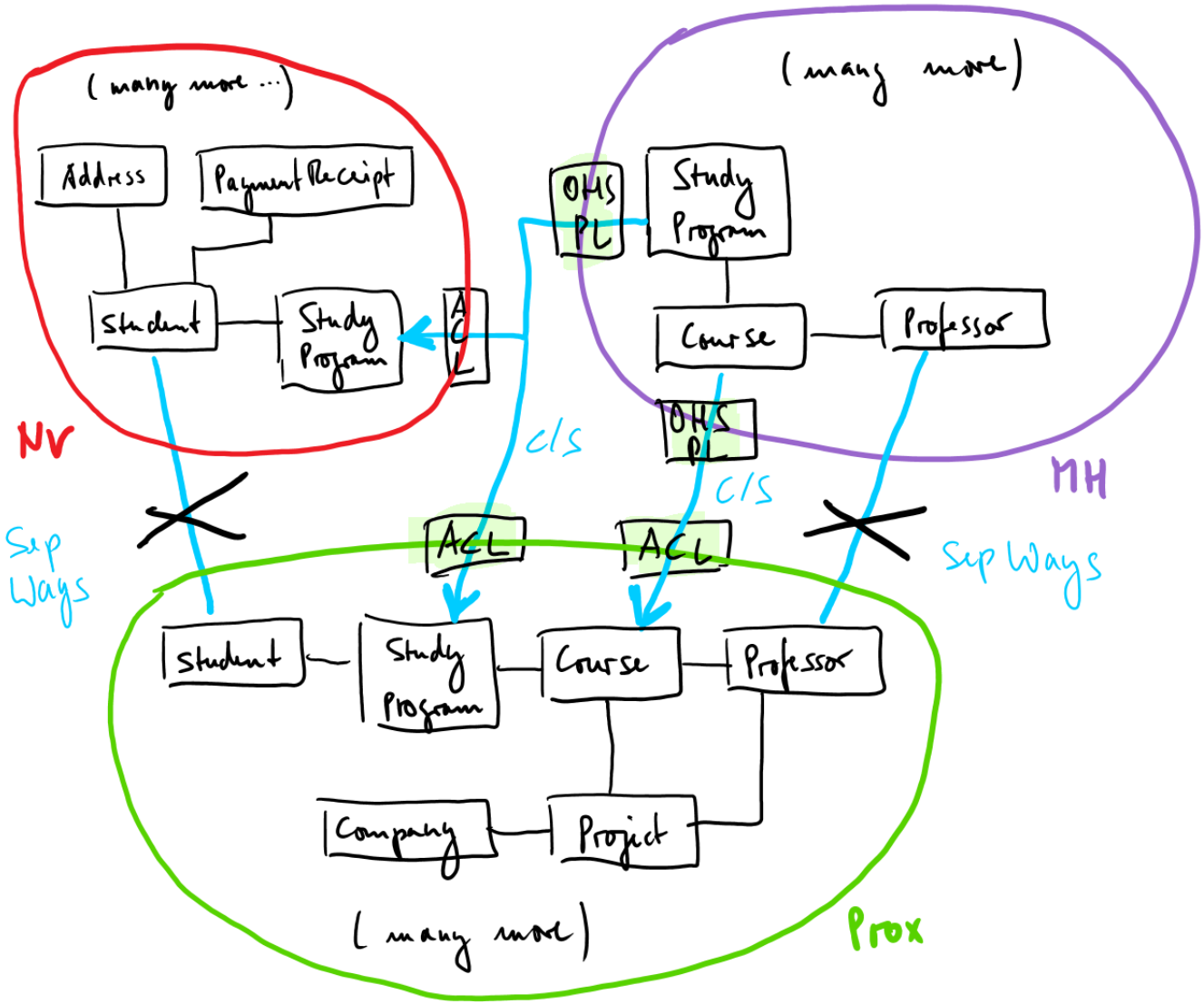
Open Host Services / Published Language / Anticorruption-Layer



- Bereitstellung eines **Open Host Service (OHS)** heißt, dass ein offen zugänglicher Service angeboten wird, um auf einen Bounded Context zugreifen zu können – z.B. wie hier ein REST-Service.
- Eine **Published Language (PL)** ist eine Spezifikation der ausgetauschten Inhalte. Hier z.B: das Verständnis darüber, was ein Student-Objekt ausmacht – also letztlich die gelieferte JSON-Struktur.

- Ein **Anticorruption Layer (ACL)** erlaubt es dem Consumer (downstream), sein eigenes Modell stabil zu halten, selbst wenn sich das Modell des Providers (upstream) sich ändert oder anders aussieht.
- Hier im (nicht sehr realistischen) Beispiel: Wenn **NV** den Student mit "firstname" und "lastname" abbildet, **Prox** aber nur den zusammengesetzten "name" kennt, dann würde der Anticorruption Layer von **Prox** die Umrechung von "firstname/lastname" in "name" vornehmen.

Annotierte Context Map für die Hochschulsysteme



- **Student:** Aus dem realen Notenverwaltungsystem wird man keine Daten beziehen können (Standardsoftware, Datenschutz, Bürokratie …). Also der Einfachheit halber *Separate Ways*.
- **Professor:** Auch hier wird im realen Fall *Separate Ways* angewendet. Ggfs. könnte man aber die Information gegen ein anderes System (Identity Management der TH Köln) abgleichen, dann aber als Conformist. (Die Campus-IT wird man nicht zu Änderungen bewegen können.)
- **Study Program** und **Course**: Sowohl Modulhandbuch (MH => zurzeit HOPS) wie auch PROX werden an der F10 entwickelt. *Customer/Supplier* ist realistisch (im Wesentlichen auch so realisiert).