

corewire

Rook-Ceph

Architektur

Folien-Hinweis

- `Space`, `Page down`: Nächste Folie
- `Page up`: Vorherige Folie
- `ESC`, `o`: Übersicht

[Zur Kapitelübersicht](#)

Ziele dieses Kapitels

Ich kann

- ✓: Aufgabe des Rook-Operators erklären
- ✓: Rolle der zentralen Ceph-Daemons benennen
- ✓: Zusammenspiel von StorageClass, PVC, PV und CSI
- ✓: beschreiben
Datenpfad vom Pod bis zum OSD nachvollziehen

Rook-Architektur

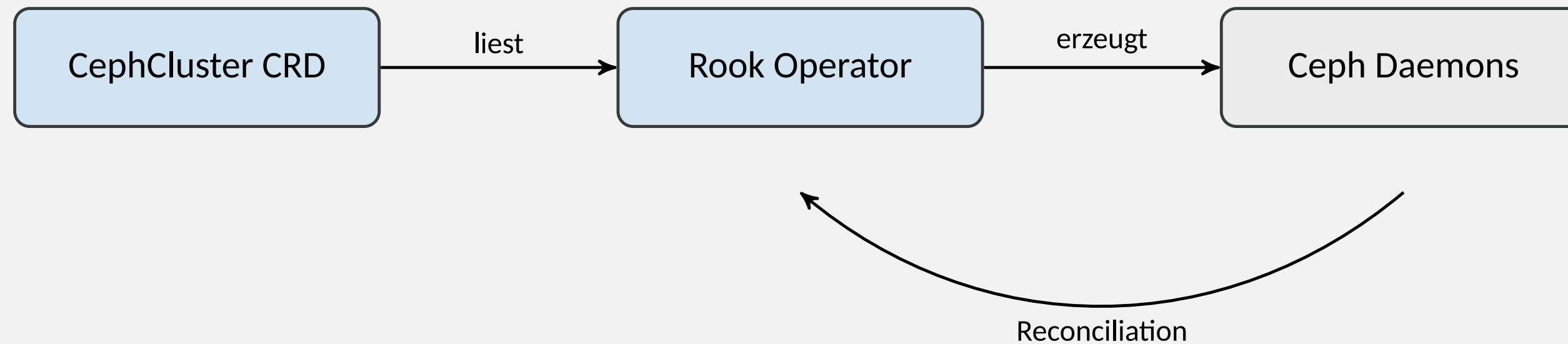
Rook als Kubernetes Operator

- Rook läuft selbst als Kubernetes-Workload.
- Er beobachtet Custom Resources wie `CephCluster` und `CephBlockPool`.
- Er übersetzt diese Ressourcen in konkrete Ceph-Konfiguration.
- Er betreibt und überwacht die Ceph-Daemons als Kubernetes-Objekte.

Desired State und Reconciliation

- CRDs beschreiben den gewünschten Zustand.
- Der Operator vergleicht Ist- und Soll-Zustand fortlaufend.
- Abweichungen werden automatisch korrigiert.
- Dieses Prinzip heißt Reconciliation.

Reconciliation-Schleife



Aufgaben des Rook-Operators

- Ceph-Daemons als Kubernetes-Objekte erzeugen und pflegen
- Konfigurationsänderungen an CRDs umsetzen
- Ausfälle erkennen und Wiederherstellung anstoßen
- Statusinformationen in Kubernetes-Ressourcen zurückspiegeln

Zusammenspiel von CRDs, Kubernetes-Objekten und Ceph-Diensten

Ebene	Beispiel
CRD	<code>CephCluster</code> , <code>CephBlockPool</code>
Kubernetes-Objekt	Deployment, Pod, ConfigMap, Secret
Ceph-Dienst	MON, MGR, OSD, MDS, RGW

Ceph-Architektur

Zentrale Ceph-Daemons

Daemon	Aufgabe
MON	Clusterzustand und Quorum
MGR	Management, Module und Metriken
OSD	Speicherung und Replikation der Daten
MDS	Metadaten für CephFS
RGW	S3-kompatibler Objektspeicher

MON – Monitor

- Speichert die Cluster-Map (Zustand von MONs, OSDs, PGs, CRUSH-Regeln).
- Mehrere MONs bilden gemeinsam ein Quorum.
- Ohne Quorum sind keine Schreib- und Lesevorgänge im Cluster möglich.
- Ungerade Anzahl an MONs für stabile Mehrheitsentscheidungen.

MGR – Manager

- Ergänzt die MONs um Management-Funktionen.
- Stellt Metriken bereit, z. B. für Prometheus.
- Betreibt Module wie Dashboard, Balancer oder Autoscaler.
- Läuft meist als Active/Standby-Paar.

OSD – Object Storage Daemon

- Läuft pro Datenträger und speichert die eigentlichen Objektdaten.
- Verantwortlich für Replikation zu anderen OSDs.
- Übernimmt Recovery und Backfill nach Ausfällen.
- Meldet eigenen Zustand an die MONs.

MDS – Metadata Server

- Wird ausschließlich für CephFS benötigt.
- Verwaltet Metadaten wie Verzeichnisstruktur und Dateiattribute.
- Nutzdaten selbst liegen weiterhin auf den OSDs.
- Kann für Skalierung und Ausfallsicherheit mehrfach betrieben werden.

RGW – RADOS Gateway

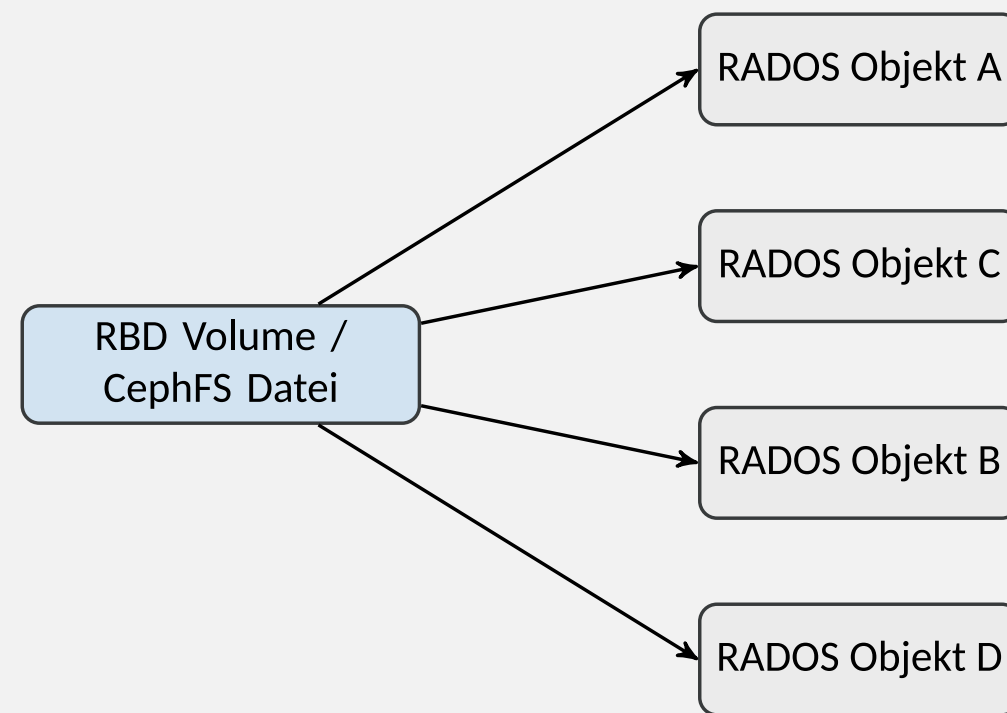
- Stellt eine S3- und Swift-kompatible API bereit.
- Übersetzt Objekt-API-Aufrufe in RADOS-Operationen auf den OSDs.
- Ermöglicht Object-Storage-Zugriff ohne eigenen Ceph-Client.
- Typisch für Backups, Artefakte und S3-basierte Anwendungen.

Rados

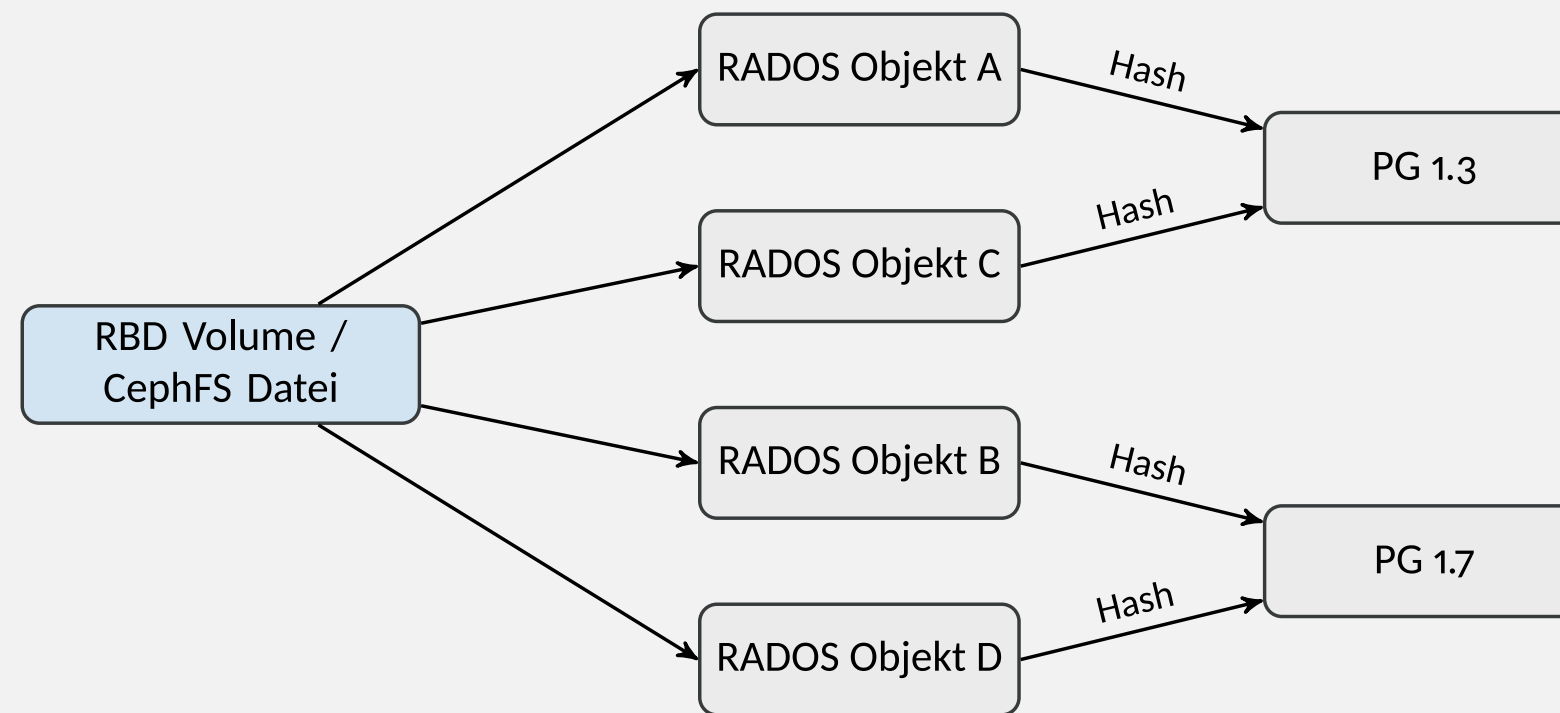
Vom Objekt zum Datenträger

RBD Volume /
CephFS Datei

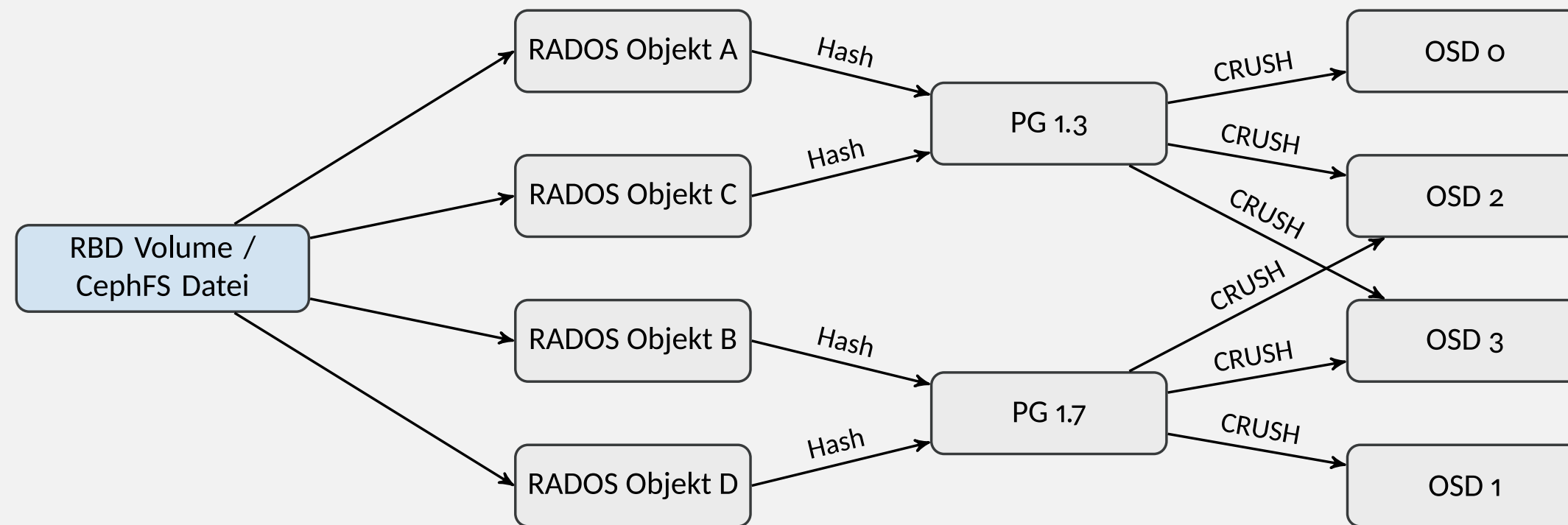
Vom Objekt zum Datenträger



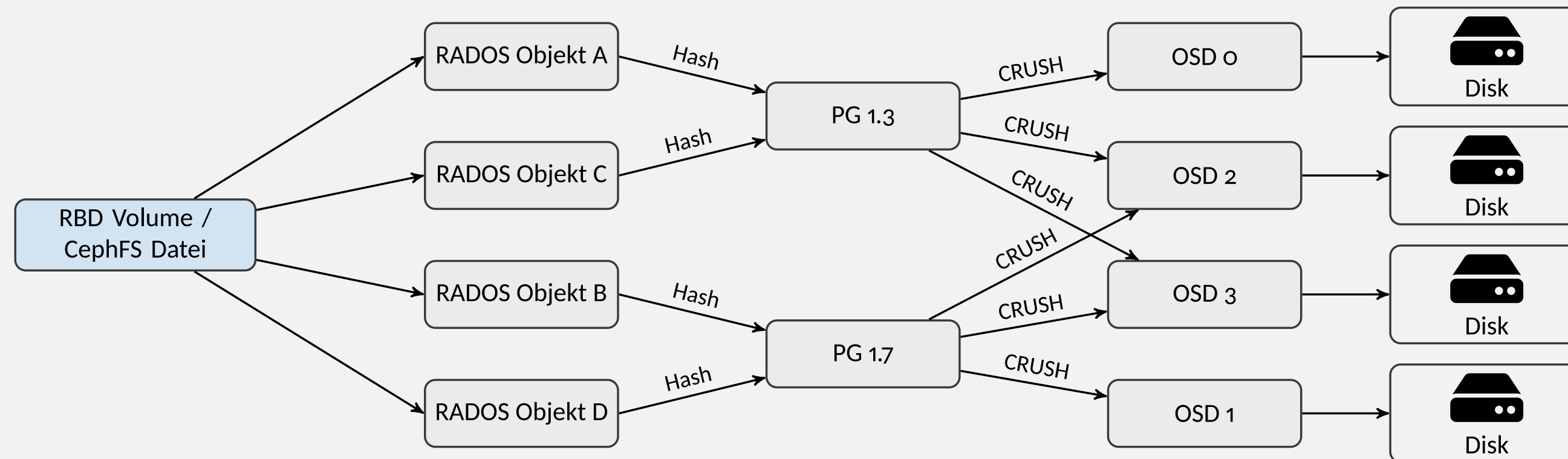
Vom Objekt zum Datenträger



Vom Objekt zum Datenträger



Vom Objekt zum Datenträger



Pools und Placement Groups

- Ein Pool ist ein logischer Container für RADOS Objekte.
- Jeder Pool wird in Placement Groups (PGs) unterteilt.
- PGs verteilen Daten auf mehrere OSDs.
- Diese Verteilung ermöglicht Parallelität und Ausfallsicherheit.

CRUSH und Failure Domains

- CRUSH berechnet, welche OSDs Daten eines Objekts speichern.
- Die Berechnung erfolgt ohne zentrale Lookup-Tabelle.
- Failure Domains legen fest, wie Kopien verteilt werden (z. B. pro Host).
- Ziel: Ein Ausfall einer Domain gefährdet nicht alle Kopien gleichzeitig.

Replikation und Erasure Coding

Verfahren	Prinzip	Trade-off
Replikation	Mehrere vollständige Kopien	Hoher Speicherbedarf, einfache Wiederherstellung
Erasure Coding	Daten- und Prüfsummen-Fragmente	Weniger Speicherbedarf, mehr CPU/Netzwerklast

Kubernetes Storage

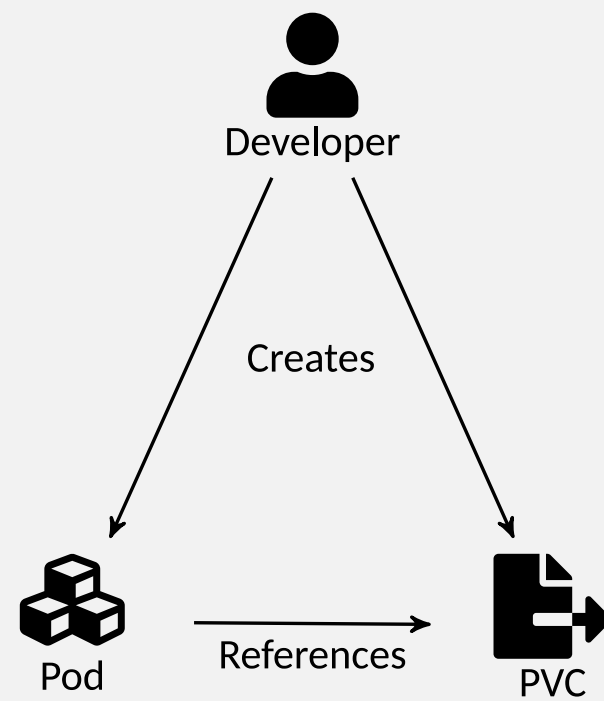
Zusammenspiel der Storage-Komponenten

- `StorageClass` beschreibt, wie ein Volume provisioniert wird.
- `PersistentVolumeClaim` fordert Speicher aus Anwendungssicht an.
- `PersistentVolume` repräsentiert den tatsächlich bereitgestellten Speicher.
- CSI Controller und CSI Node Plugin setzen Provisionierung und Mounting um.
- Am Ende steht ein Ceph Pool, in dem die Daten liegen.

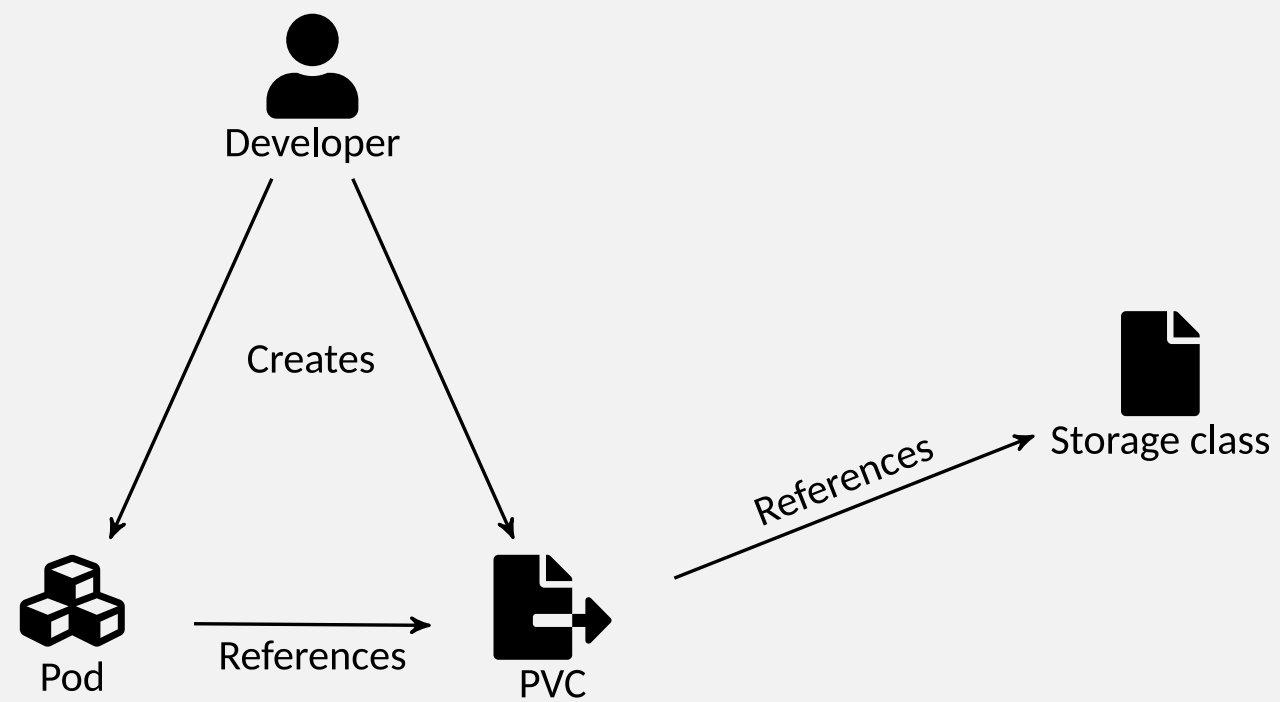
CSI als Verbindungsstelle



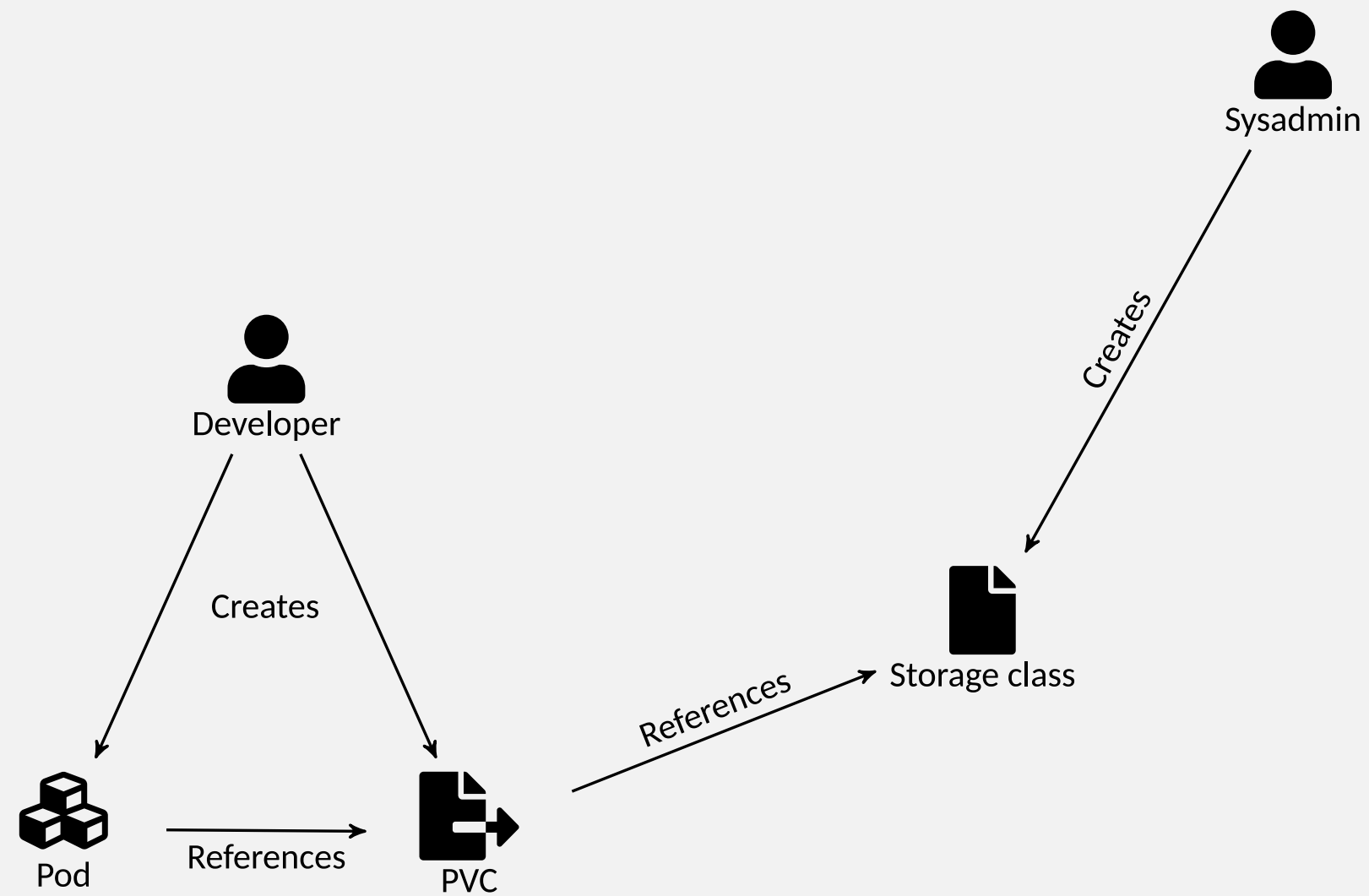
CSI als Verbindungsstelle



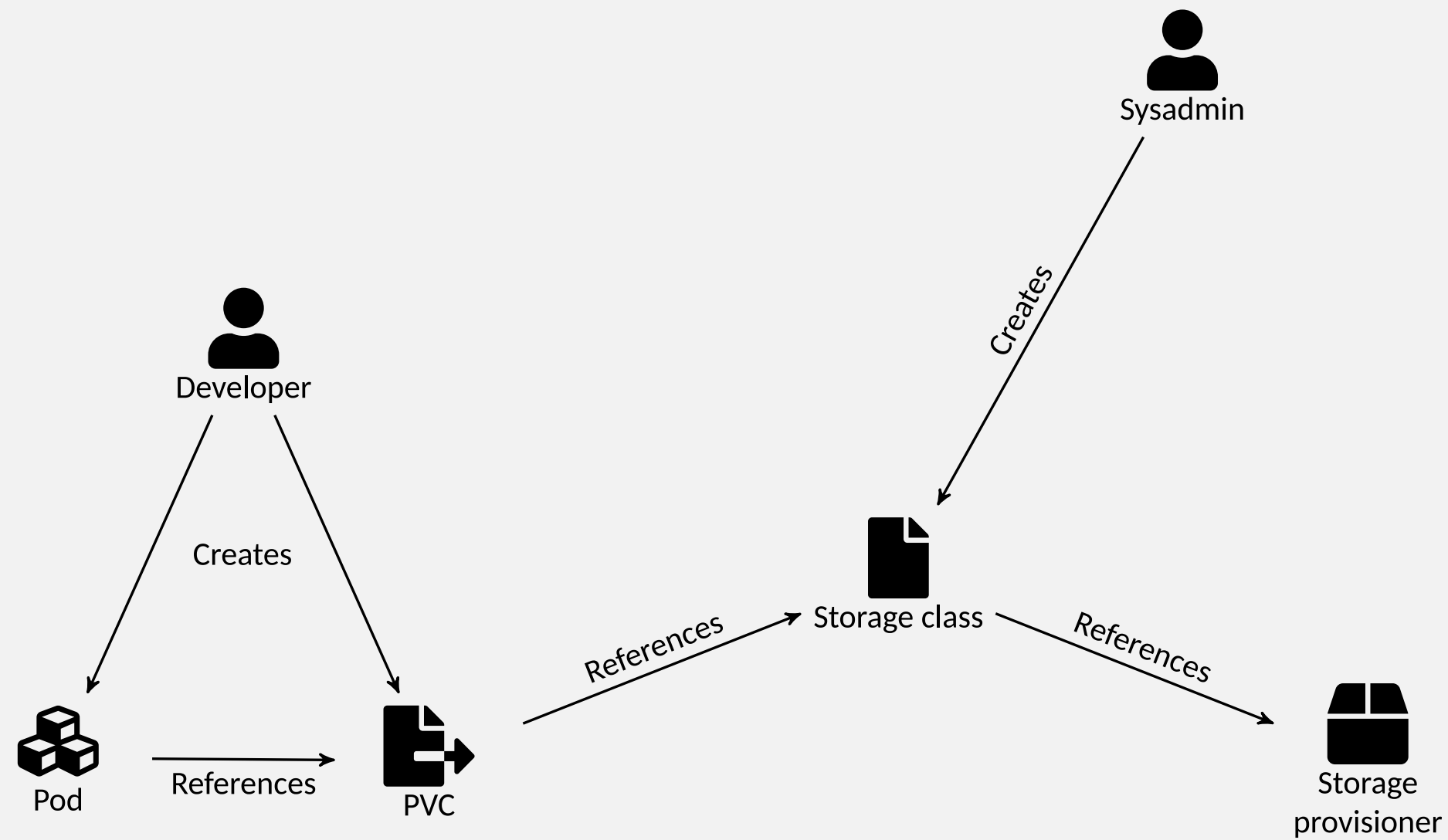
CSI als Verbindungsstelle



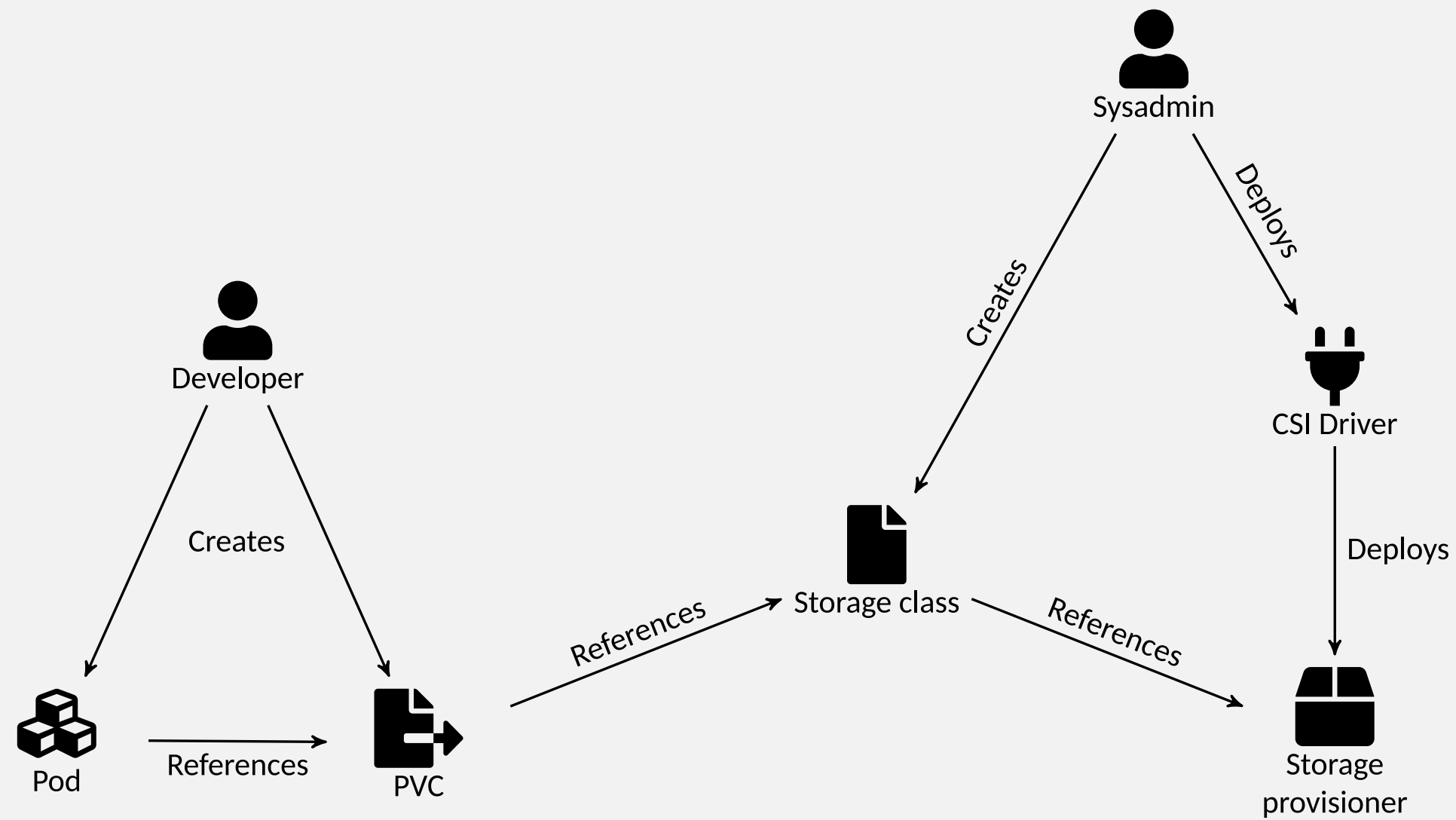
CSI als Verbindungsstelle



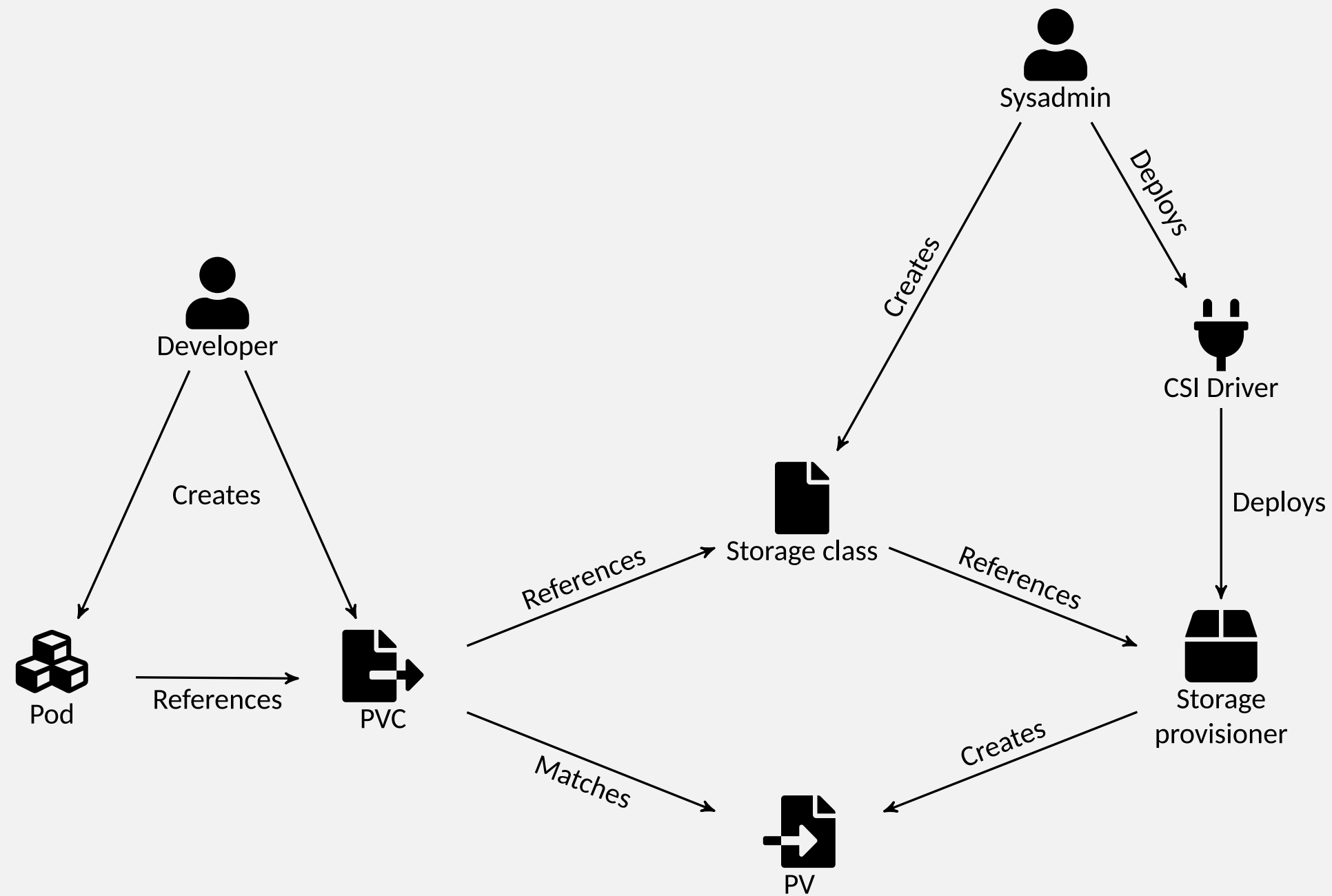
CSI als Verbindungsstelle



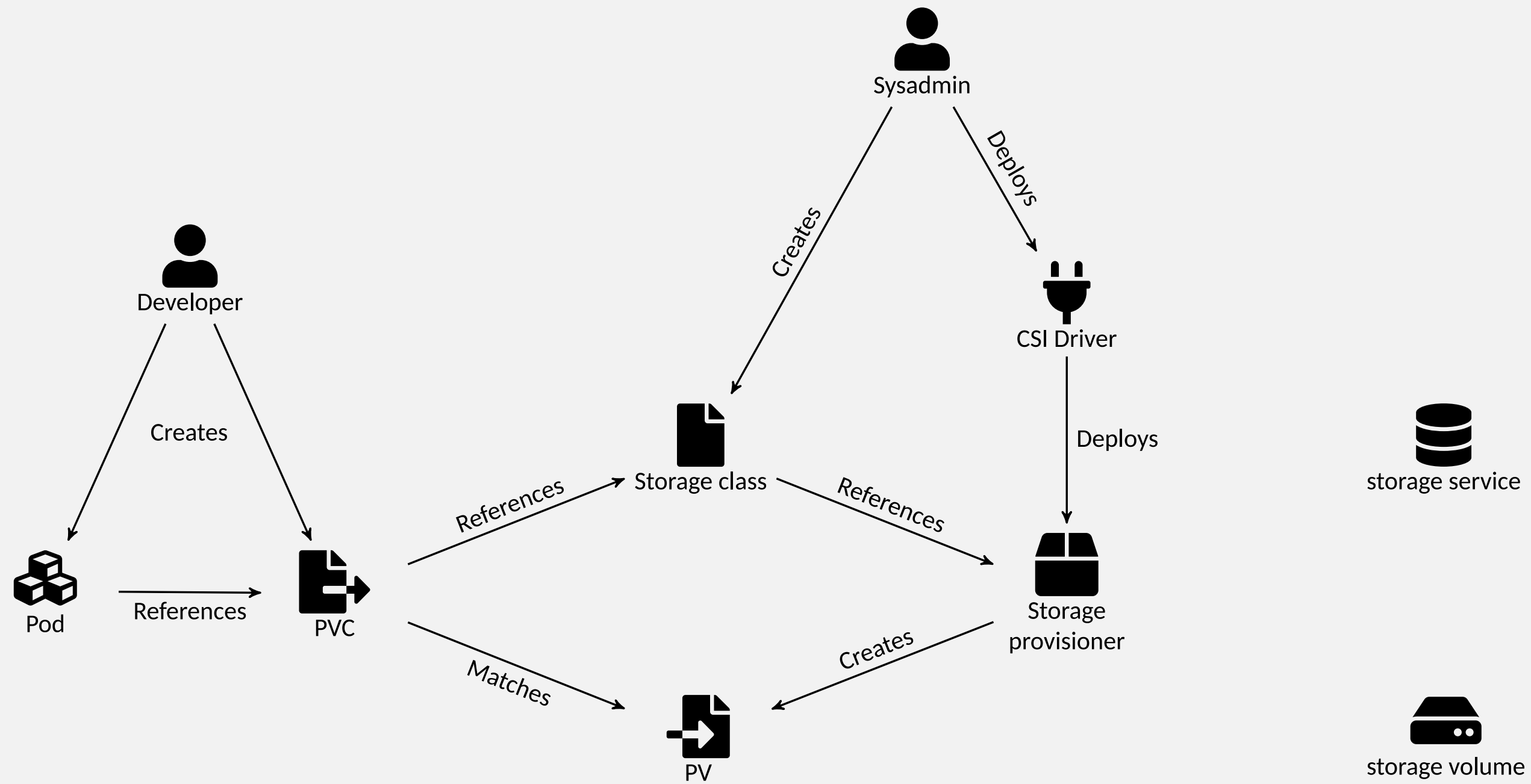
CSI als Verbindungsstelle



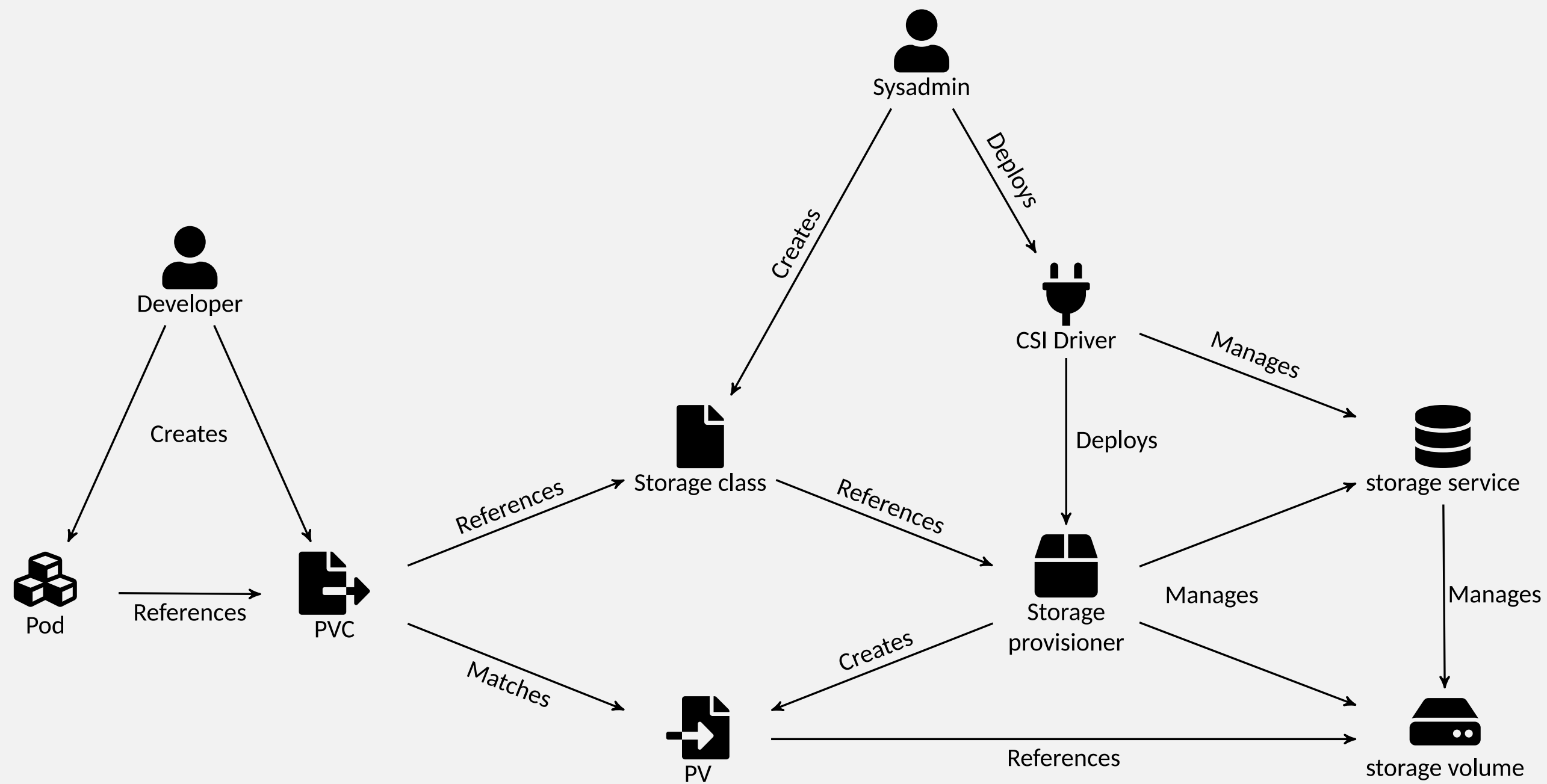
CSI als Verbindungsstelle



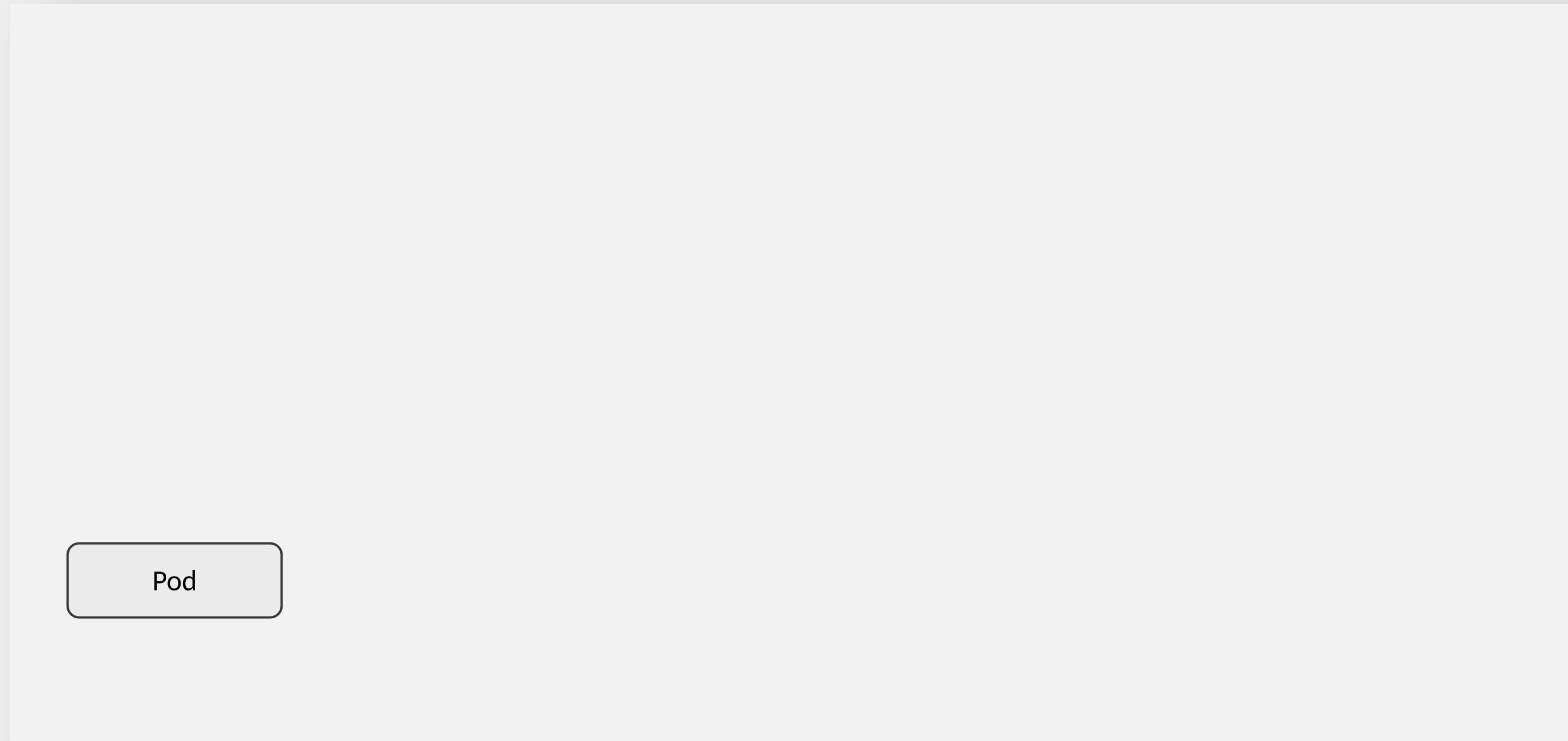
CSI als Verbindungsstelle



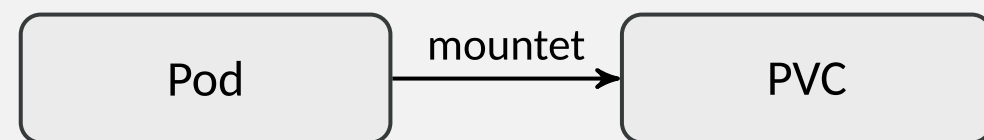
CSI als Verbindungsstelle



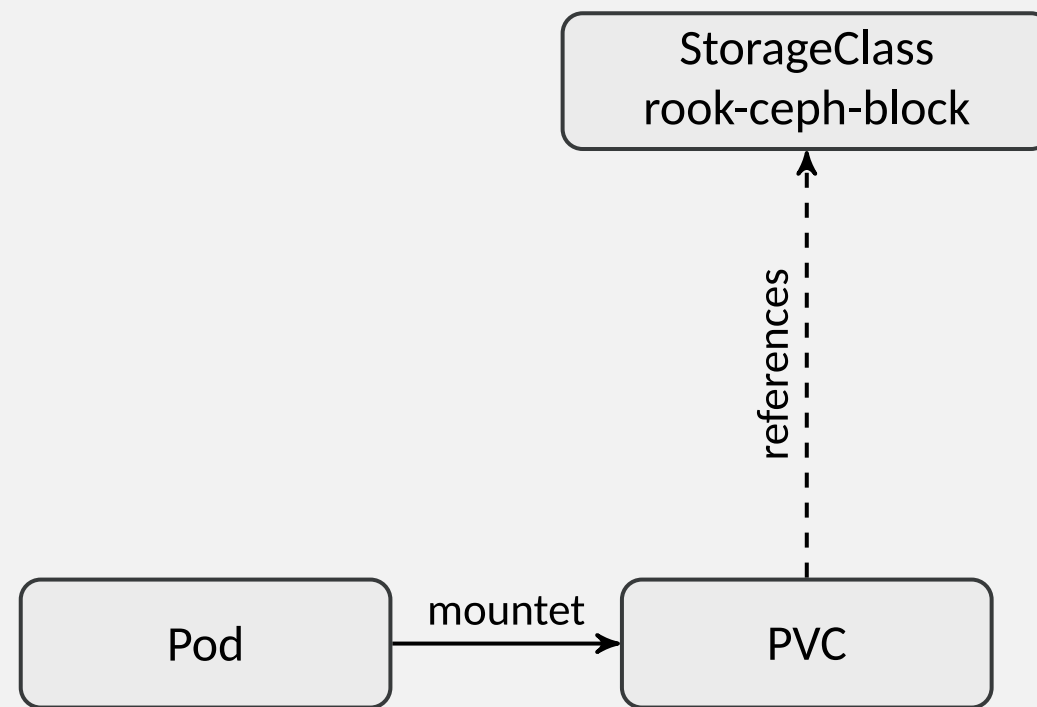
Von der PVC zum RBD Image



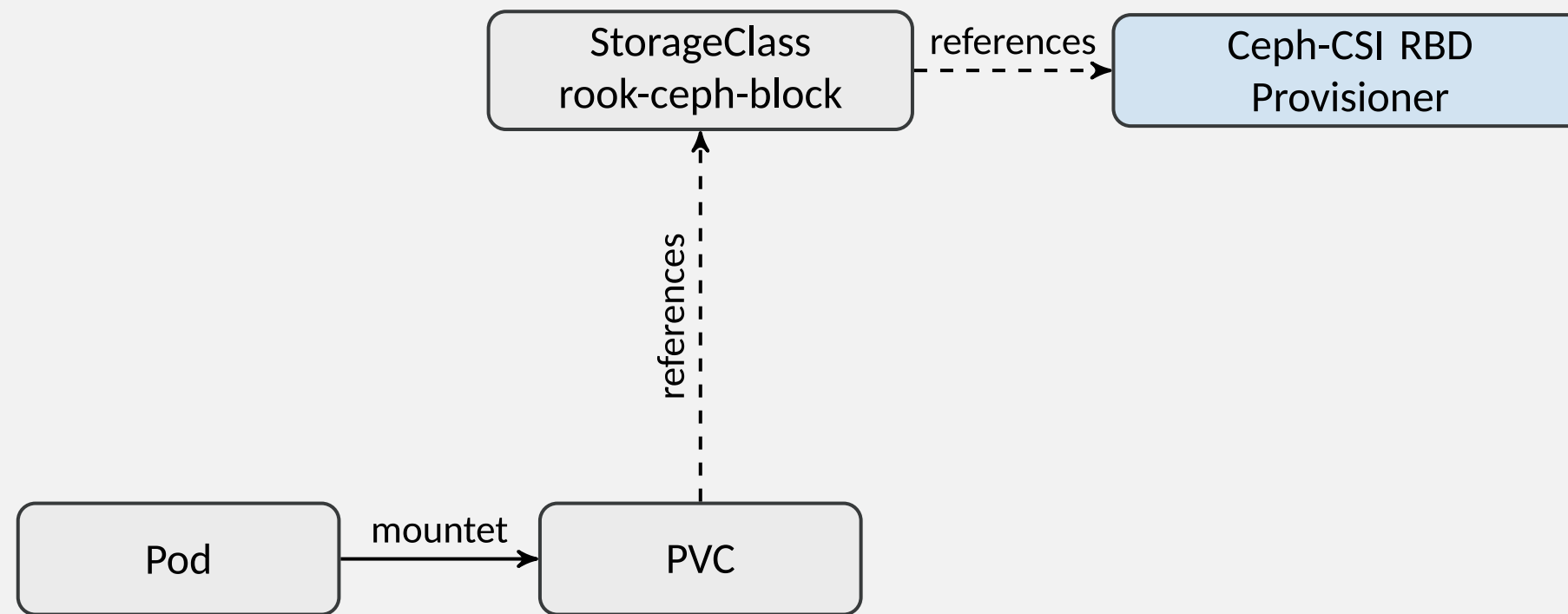
Von der PVC zum RBD Image



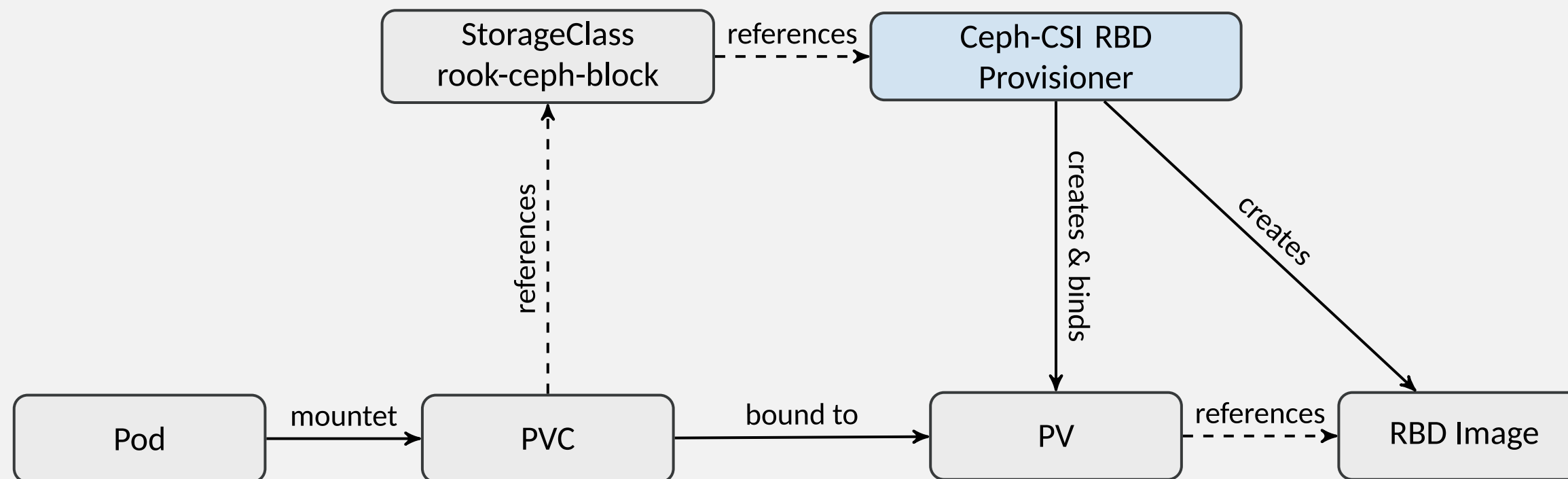
Von der PVC zum RBD Image



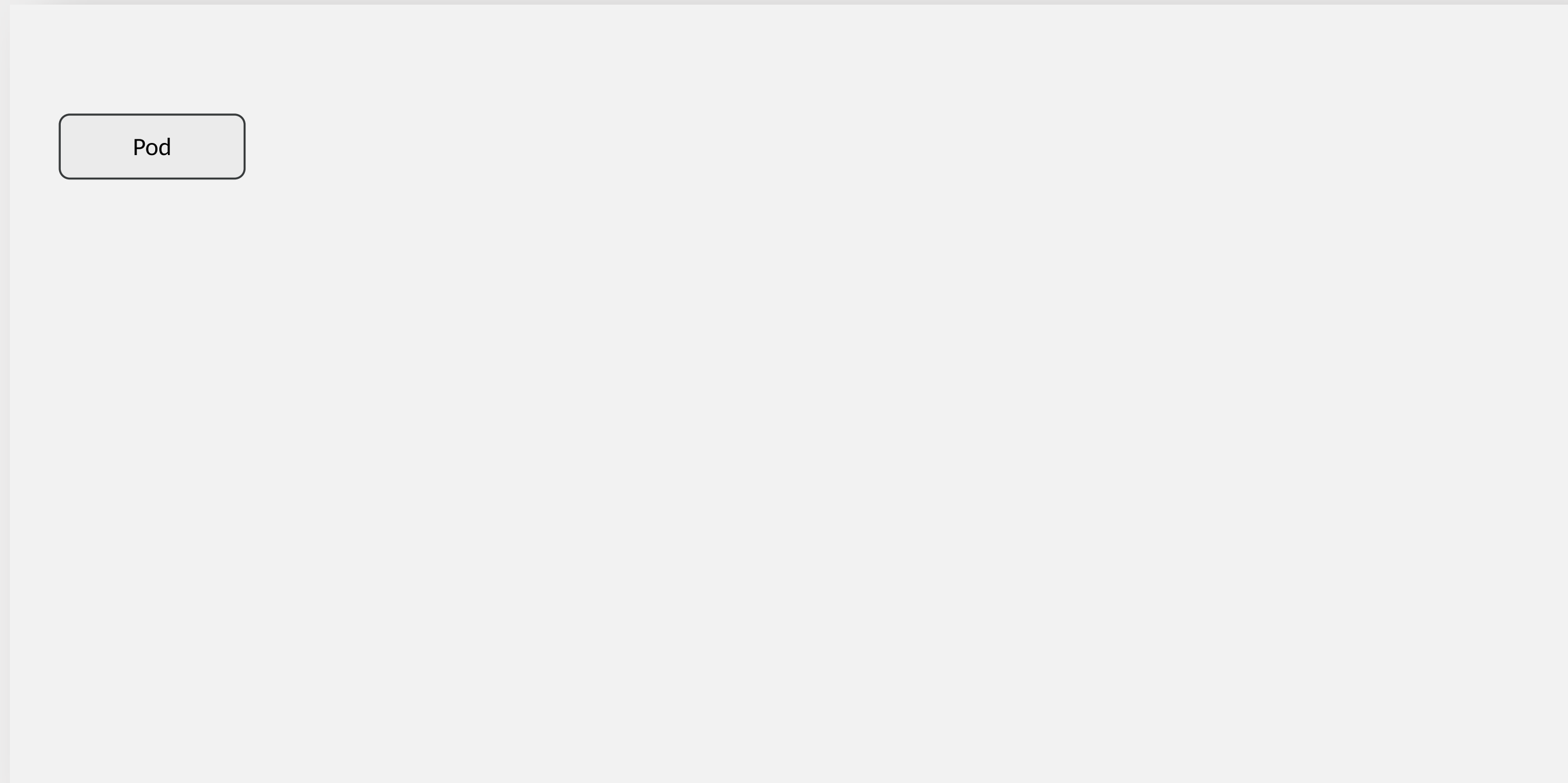
Von der PVC zum RBD Image



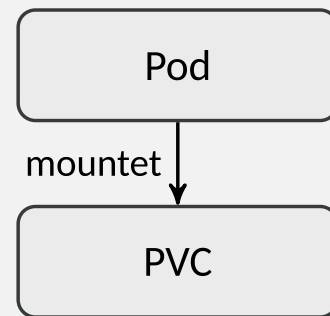
Von der PVC zum RBD Image



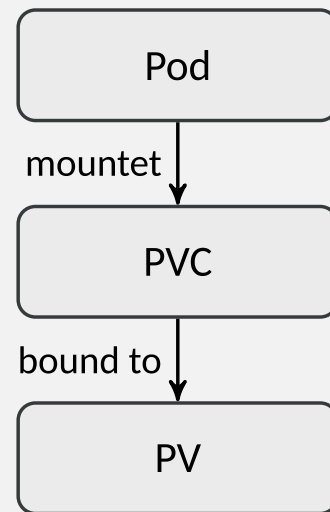
Datenpfad vom Pod bis zum OSD



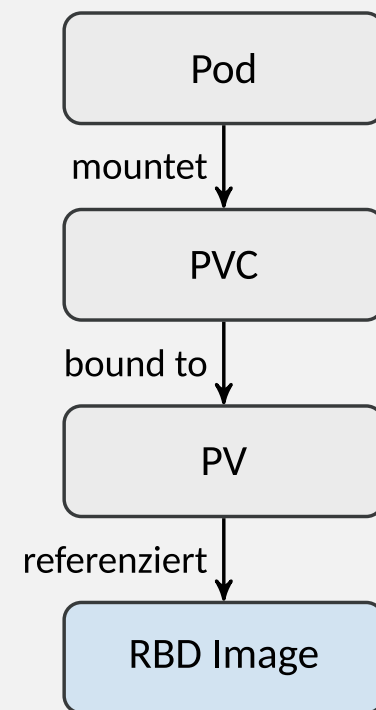
Datenpfad vom Pod bis zum OSD



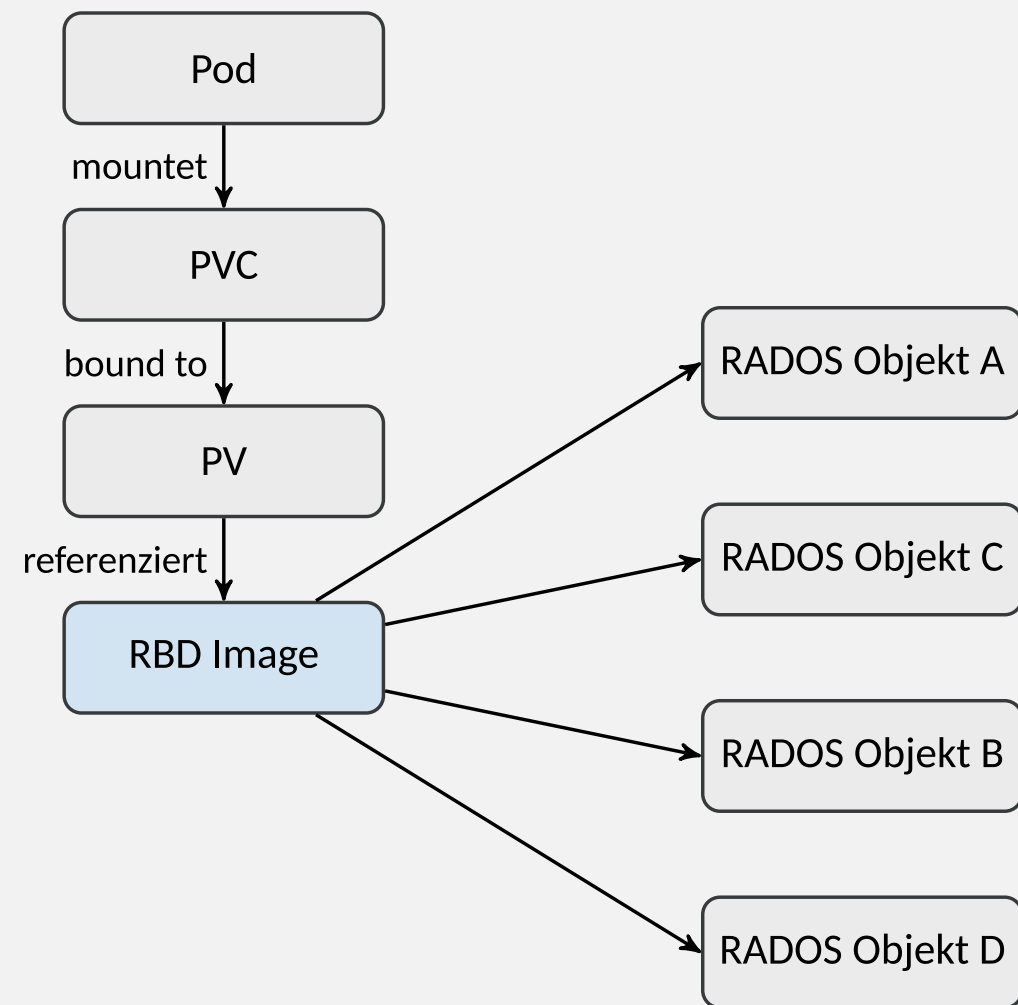
Datenpfad vom Pod bis zum OSD



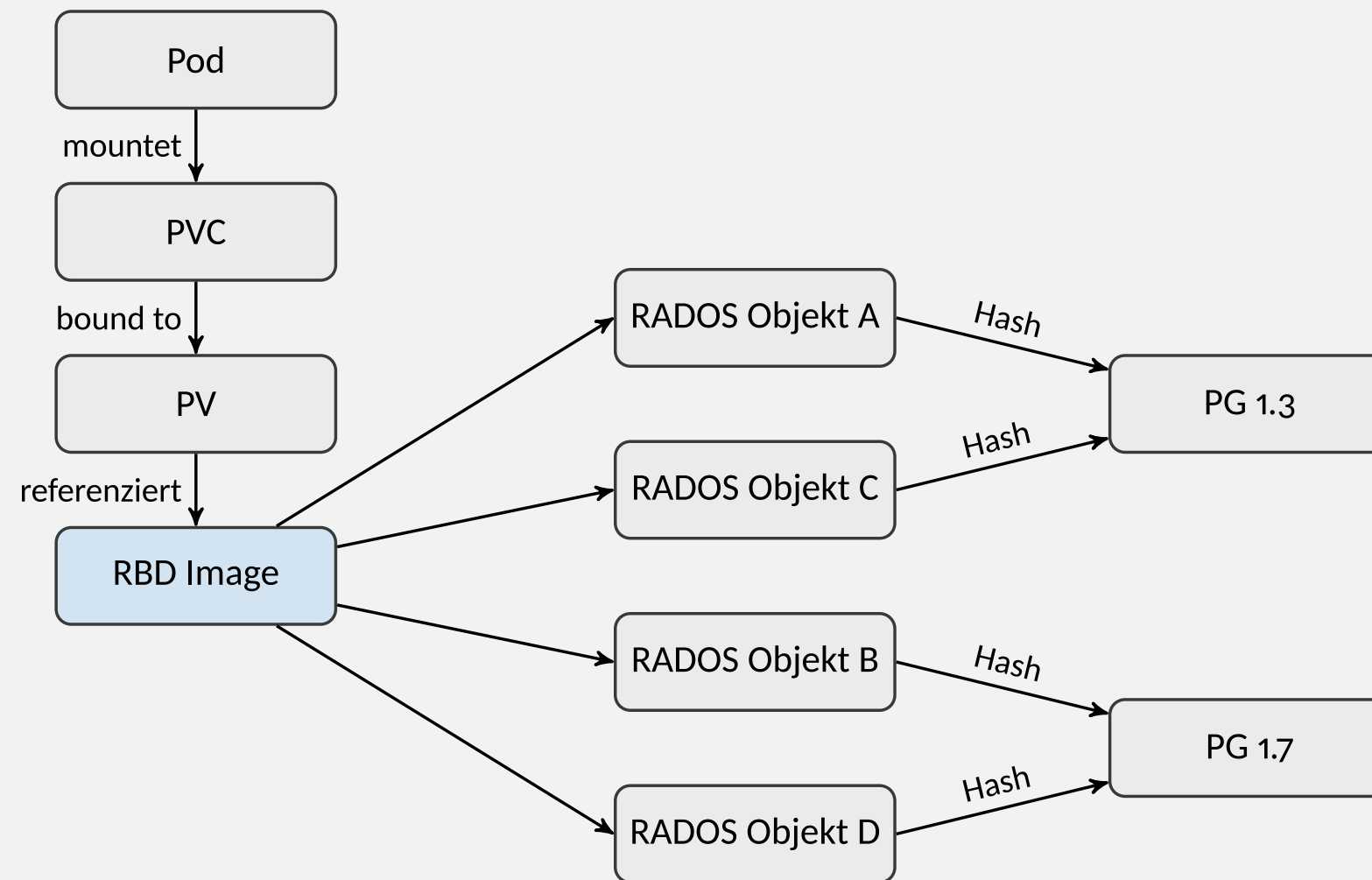
Datenpfad vom Pod bis zum OSD



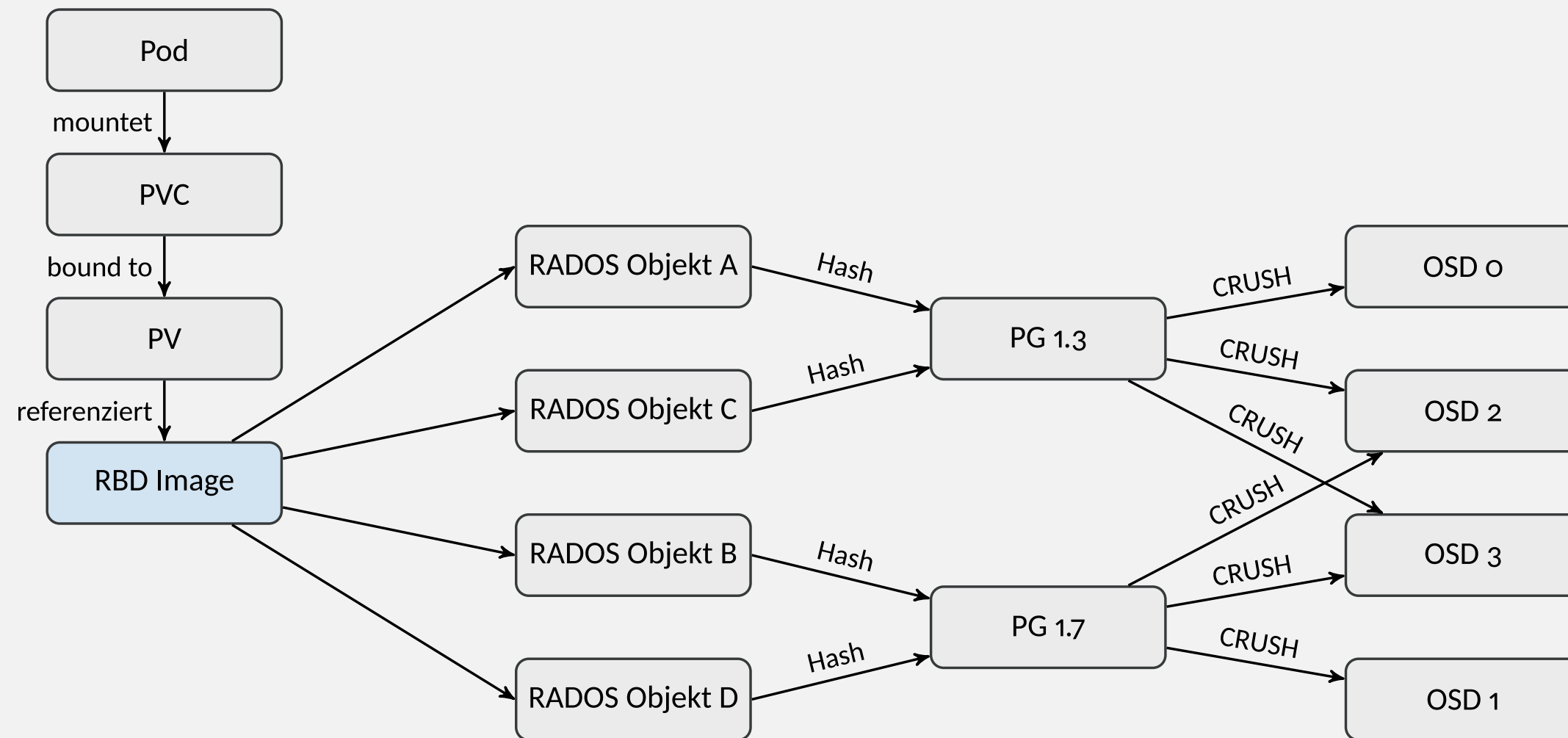
Datenpfad vom Pod bis zum OSD



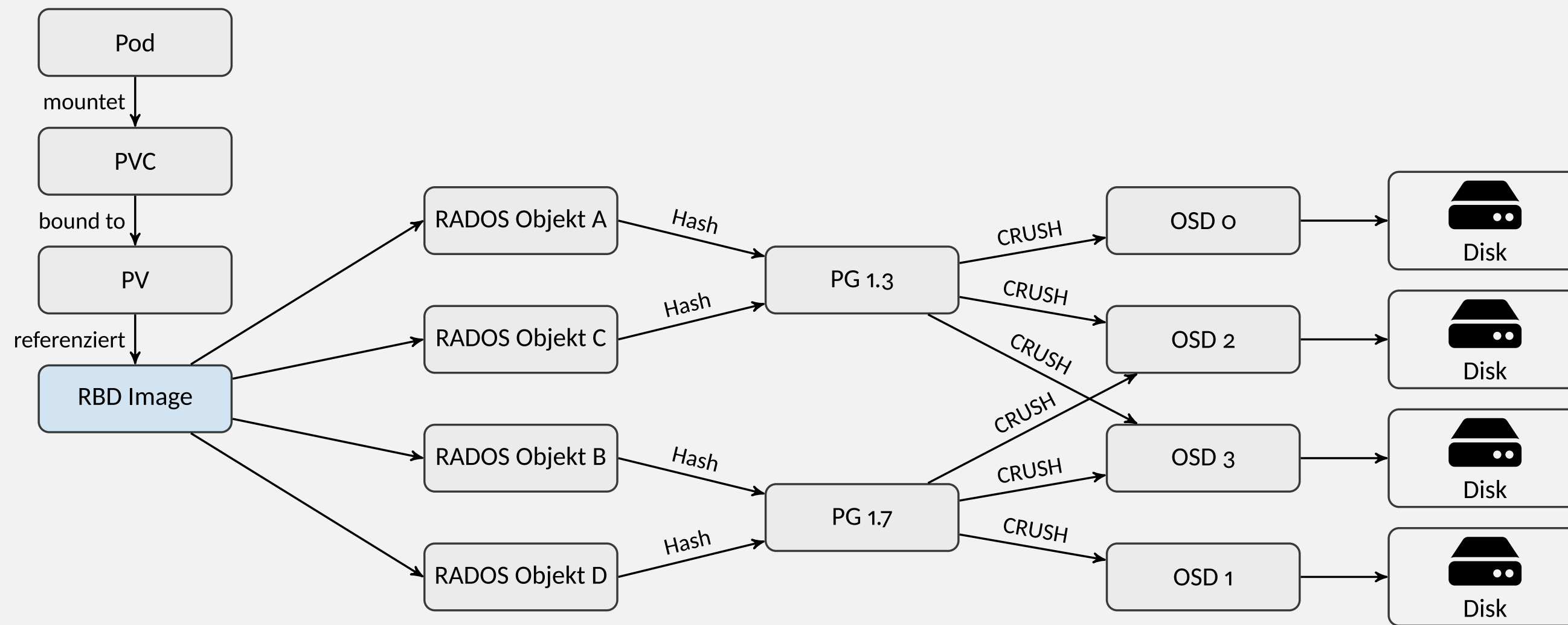
Datenpfad vom Pod bis zum OSD



Datenpfad vom Pod bis zum OSD



Datenpfad vom Pod bis zum OSD



Demo: CephFS

Wichtige Befehle

Kubernetes-Ebene

StorageClasses auflisten

```
kubectl get storageclass
```

Provisioner, Pool und weitere Parameter einer StorageClass anzeigen.

```
kubectl get storageclass <SC> -o yaml
```

Ceph-Ebene (Rook-Toolbox)

Rook-Toolbox öffnen

```
kubectl exec -it deploy/rook-ceph-tools -- bash
```

Die Rook-Toolbox enthält alle wichtigen Ceph-Kommandos, ist gegen den Cluster authentifiziert und ist im Namespace `rook-ceph` verfügbar.

Clusterzustand prüfen

Allgemeiner Status des Ceph-Clusters

```
ceph status
```

Pools auflisten

```
ceph osd pool ls
```

RBD-Images analysieren

RBD-Images im Pool auflisten

```
rbd ls <POOL>
```

RBD-Image-Details anzeigen

```
rbd info <POOL>/<IMAGE>
```

RBD-Image-Metadaten anzeigen

```
rbd image-meta list <POOL>/<IMAGE>
```

RADOS-Ebene

RADOS-Objekte

Alle RADOS-Objekte im Pool auflisten

```
rados -p <POOL> ls
```

RADOS-Objekte filtern, z. B. nach dem Namen des RBD-Images

```
rados -p <POOL> ls | grep <IMAGE>
```

Inhalt eines RADOS-Objekts auslesen

```
rados -p <POOL> get <OBJEKT> -
```

Lab

- Gehen Sie auf: <https://learn.corewire.de>
- Navigieren Sie nach:
 - Rook
 - Aufgaben
 - Architektur

Zur Kapitelübersicht

- Vorheriges Kapitel: [Einführung](#)
- Nächstes Kapitel: [Konfiguration](#)