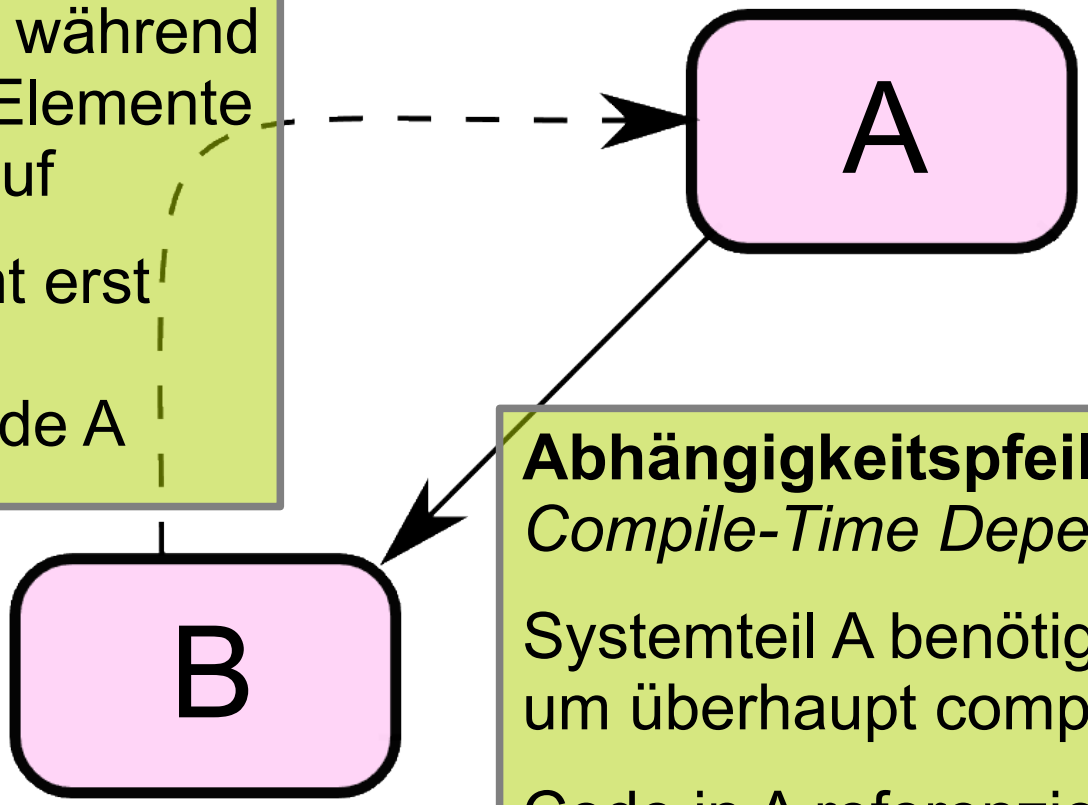
Die Bedeutung der Pfeile

Aufrufpfeil
Runtime Dependency

Systemteil B ruft während der Ausführung Elemente in Systemteil A auf

Code B bekommt erst zur Laufzeit eine Referenz auf Code A



Abhängigkeitspfeil
Compile-Time Dependency

Systemteil A benötigt Systemteil B, um überhaupt compilieren zu können

Code in A referenziert den Code in B direkt mit Namen

Abhängigkeiten gestalten

- Software-Architektur ist die Kunst, die Abhängigkeiten zwischen Systemteilen willentlich und zum Vorteil der Beteiligten zu gestalten
- Die Richtung und Art der Pfeile im Architektur-diagramm festzulegen, ist die Aufgabe des Software-Architekten
- Die Richtung kann beliebig gewählt werden
- Wir können die Richtung jederzeit umdrehen!

Inversion of Control

A

```
public class Schalter {
    private final Lampe lampe;
    private boolean gedrueckt;

    public Schalter(final Lampe lampe) {
        this.lampe = lampe;
    }

    public void drueckeSchalter() {
        if (this.gedrueckt) {
            lampe.ausschalten();
            this.gedrueckt = false;
            return;
        }
        lampe.anschalten();
        this.gedrueckt = true;
    }
}
```

B

```
public class Lampe {

    private boolean leuchtet = false;

    public void anschalten() {
        this.leuchtet = true;
    }

    public void ausschalten() {
        this.leuchtet = false;
    }
}
```

Inversion of Control

A

```
public class Schalter {
    private final Schaltbar lampe;
    private boolean gedrueckt;

    public Schalter(Schaltbar lampe) {
        this.lampe = lampe;
    }

    public void drueckeSchalter() {
        if (this.gedrueckt) {
            lampe.ausschalten();
            this.gedrueckt = false;
            return;
        }
        lampe.anschalten();
        this.gedrueckt = true;
    }
}
```

```
public interface Schaltbar {
    public void ausschalten();

    public void anschalten();
}
```

```
public class Lampe
                implements Schaltbar {

    private boolean leuchtet = false;

    @Override
    public void anschalten() {
        this.leuchtet = true;
    }

    @Override
    public void ausschalten() {
        this.leuchtet = false;
    }
}
```

Inversion of Control

```
public interface Schaltbar {
    public void ausschalten();

    public void anschalten();
}
```

```
public class Schalter {
    private final Schaltbar lampe;
    private boolean gedrueckt;

    public Schalter(Schaltbar lampe) {
        this.lampe = lampe;
    }

    public void drueckeSchalter() {
        if (this.gedrueckt) {
            lampe.ausschalten();
            this.gedrueckt = false;
            return;
        }
        lampe.anschalten();
        this.gedrueckt = true;
    }
}
```

B

```
public class Lampe
                implements Schaltbar {

    private boolean leuchtet = false;

    @Override
    public void anschalten() {
        this.leuchtet = true;
    }

    @Override
    public void ausschalten() {
        this.leuchtet = false;
    }
}
```