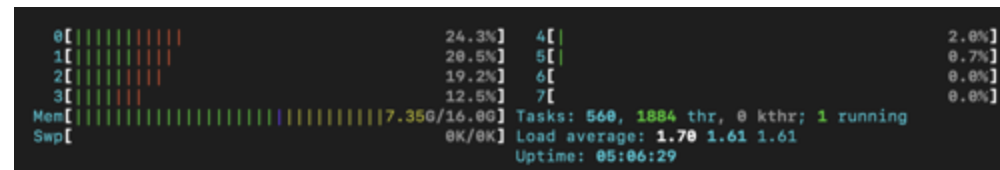


Einführung

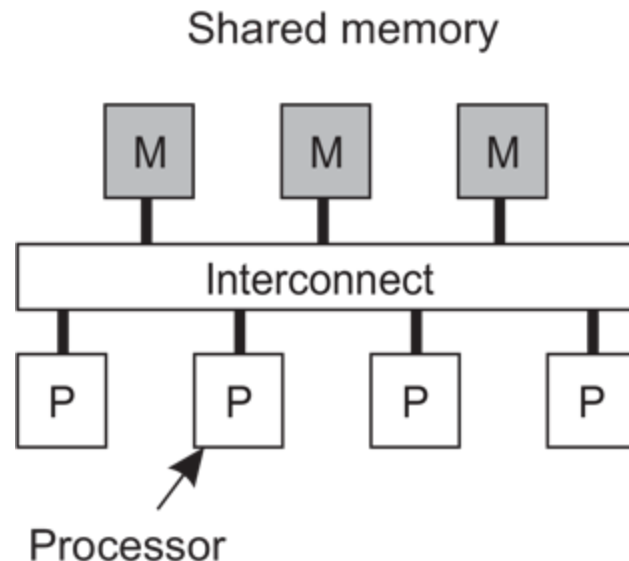
«Parallele Systeme»

- Eine single core CPU kann nur einen Prozess gleichzeitig ausführen
- Multi-core CPUs entsprechend mehrere gleichzeitig
- Ausser in sehr einfachen Embedded Systemen müssen jedoch immer sehr viele Prozesse «gleichzeitig» ausgeführt werden
können z.B. auf einem Server oder auf einem Desktop Computer

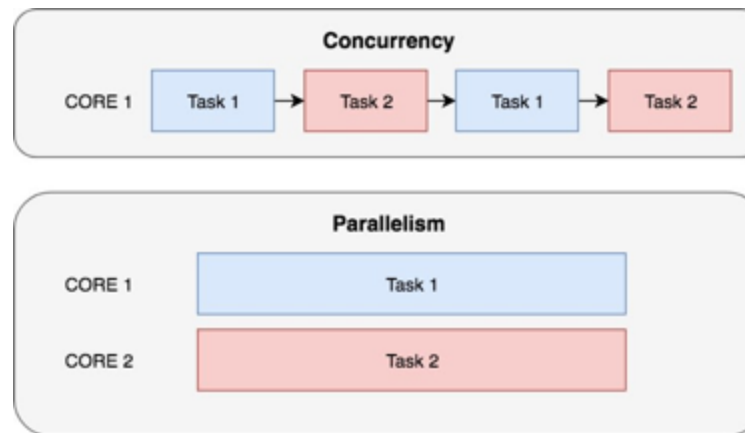


«Parallele Systeme»

- Viele verschiedene Prozesse (tausende) werden von einem oder mehreren (bis zu dutzenden) Prozessoren ausgeführt
- Ein einzelner Prozessor kann demnach nacheinander mehrere Prozesse bearbeiten
- Die Prozessoren befinden sich auf demselben Chip oder auf dem selben Mainboard
- Sie haben geteilten sowie gemeinsamen Speicher
- Die Verbindung zwischen ihnen (Interconnect) hat geringe Latenz, hohe Bandbreite und ist zuverlässig.



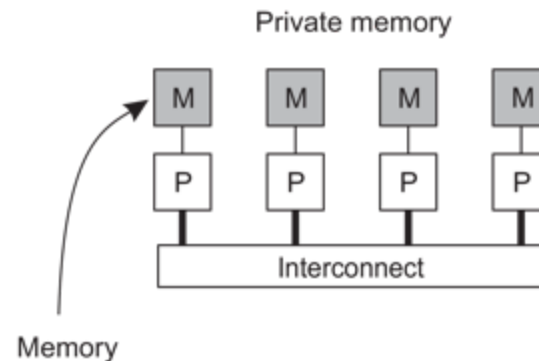
- Parallele Ausführung (parallelism): Mehr als eine Aufgabe wird gleichzeitig ausgeführt
- Nebenläufig (concurrency): Mehr als eine Aufgabe wird abgearbeitet (durch schnelles context switching)



- Eine zentrale Aufgabe von Betriebssystemen ist es, die Prozesse auf die CPUs zu verteilen.
- Dies wird «Scheduling» genannt.

Verteilte Systeme

«A distributed system is a collection of independent computers that appears to its users as a single coherent system.»



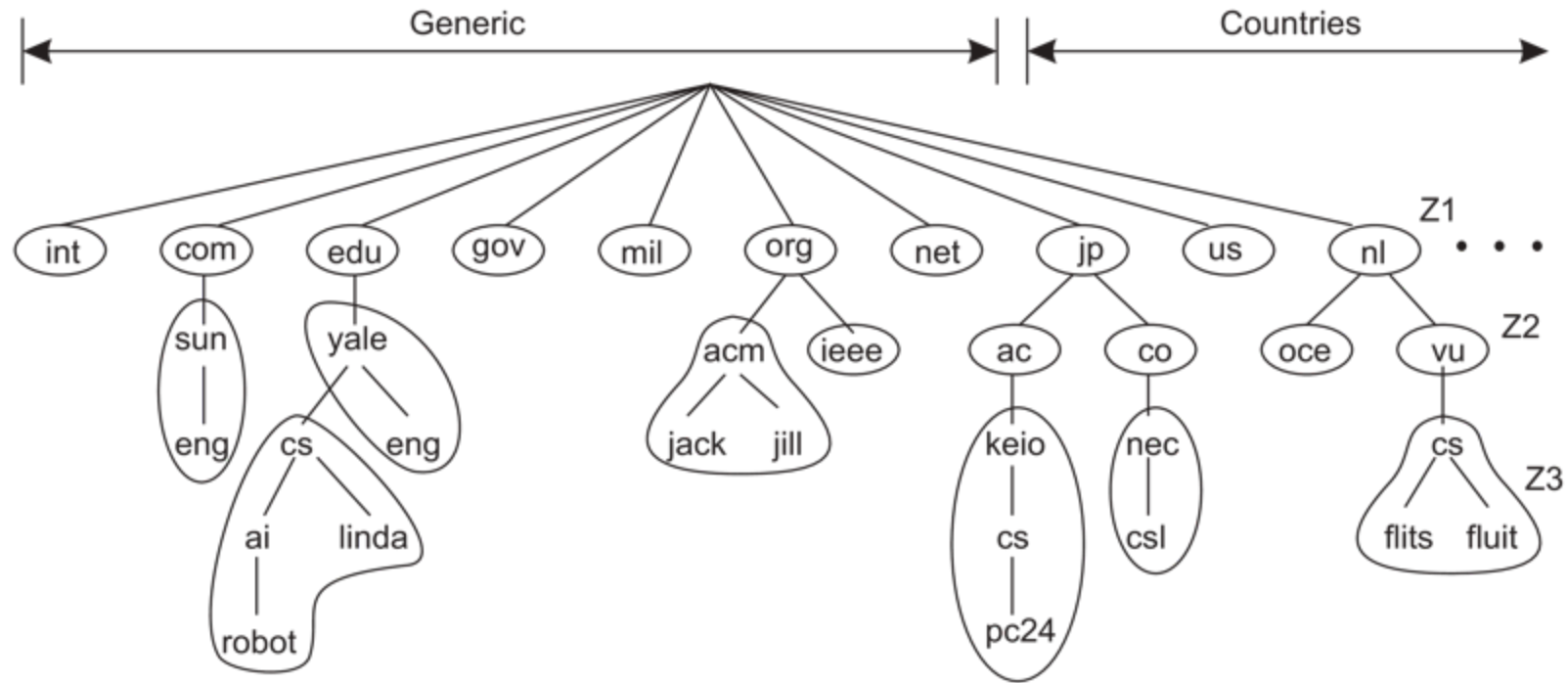
VanSteen, 2017, S. 26

P: Prozessor, Interconnect: Netzwerkverbindung, meistens HTTP, UDP/TCP, IP, Ethernet basiert

Resource Sharing

- Ressourcen verfügbar machen: Drucker, Computing, Storage, Daten, Netzwerk
- Teure Ressourcen können besser ausgelastet werden und müssen nicht mehrfach angeschafft werden
- Zusammenarbeit

Domain Name System



When the cloud leaves the datacenter

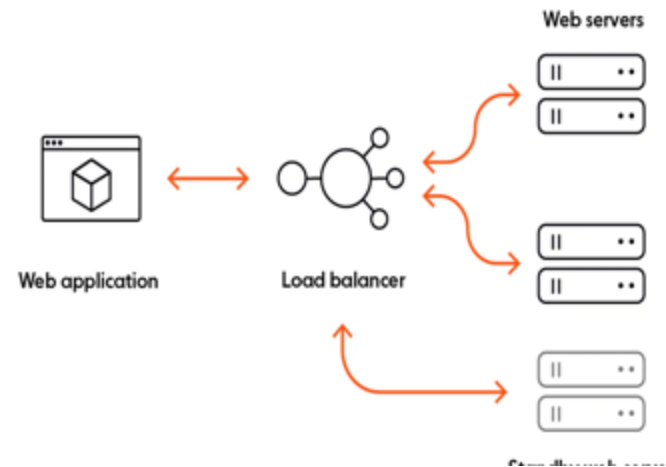


Anforderungen an moderne Software

- Hohe Verfügbarkeit
- Skalierbarkeit
- Im Katastrophenfall sollen die Systeme schnell wiederhergestellt werden können
- Soll funktionieren, auch wenn Teile des Systems Offline sind (Resilienz)
- Kostengünstig
- Einfach
- Updates müssen einfach eingespielt werden können

Lösungsansätze

- Replication: Masking Failures
- Tradeoff: Teuer und Komplex

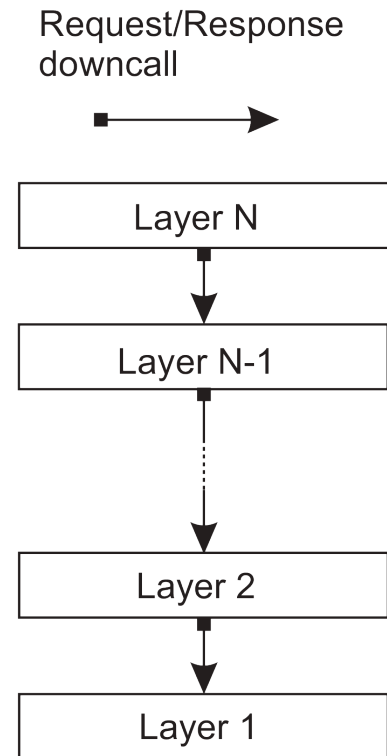


Populäre verteilte Systeme

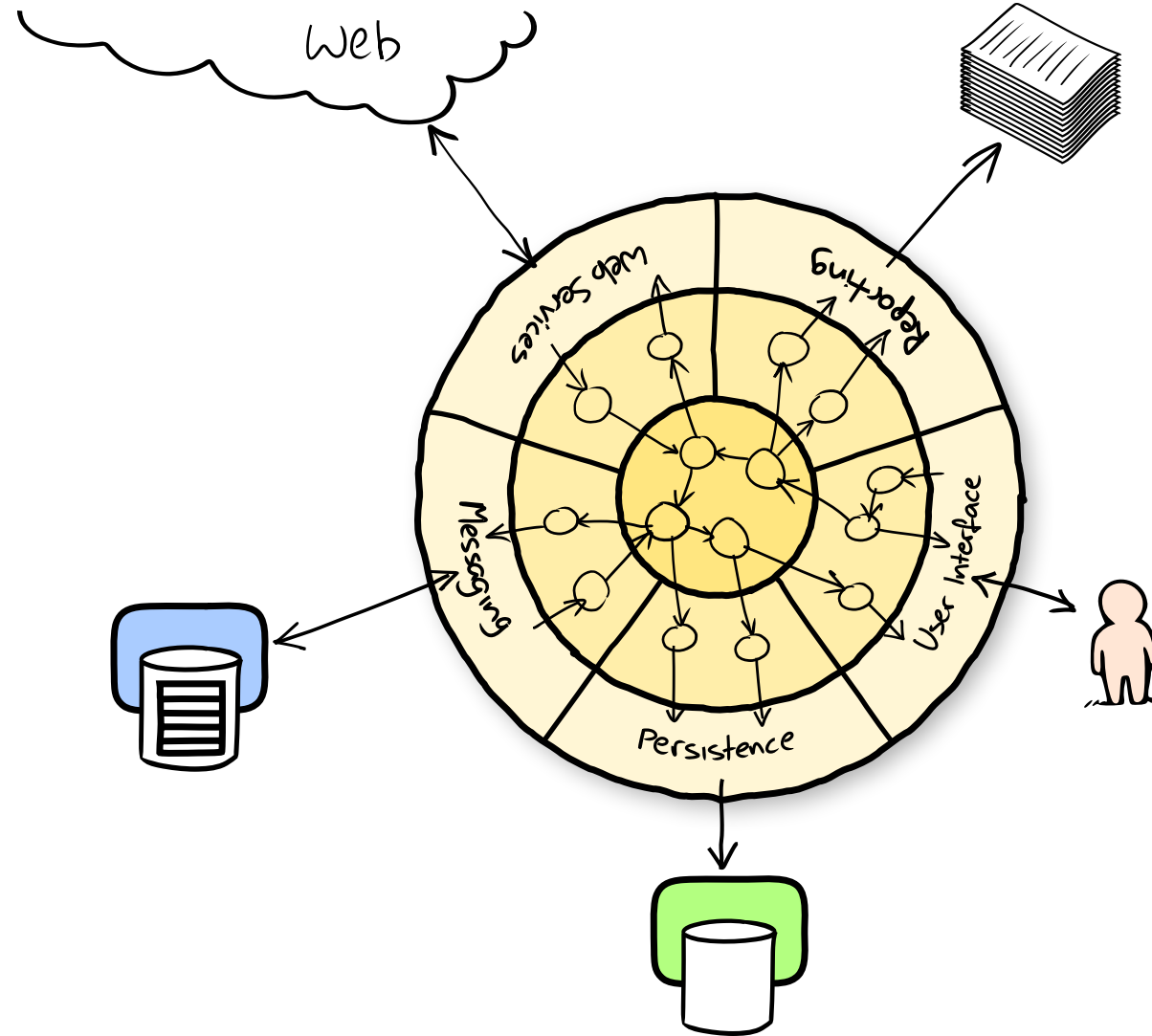
- [Matrix](#)
- [Mastodon](#)
- [Nextcloud](#)
- [CockroachDB](#)
- [Neon](#)
- [Abyl](#)
- ...

Architekturen

Schichtenarchitekturen

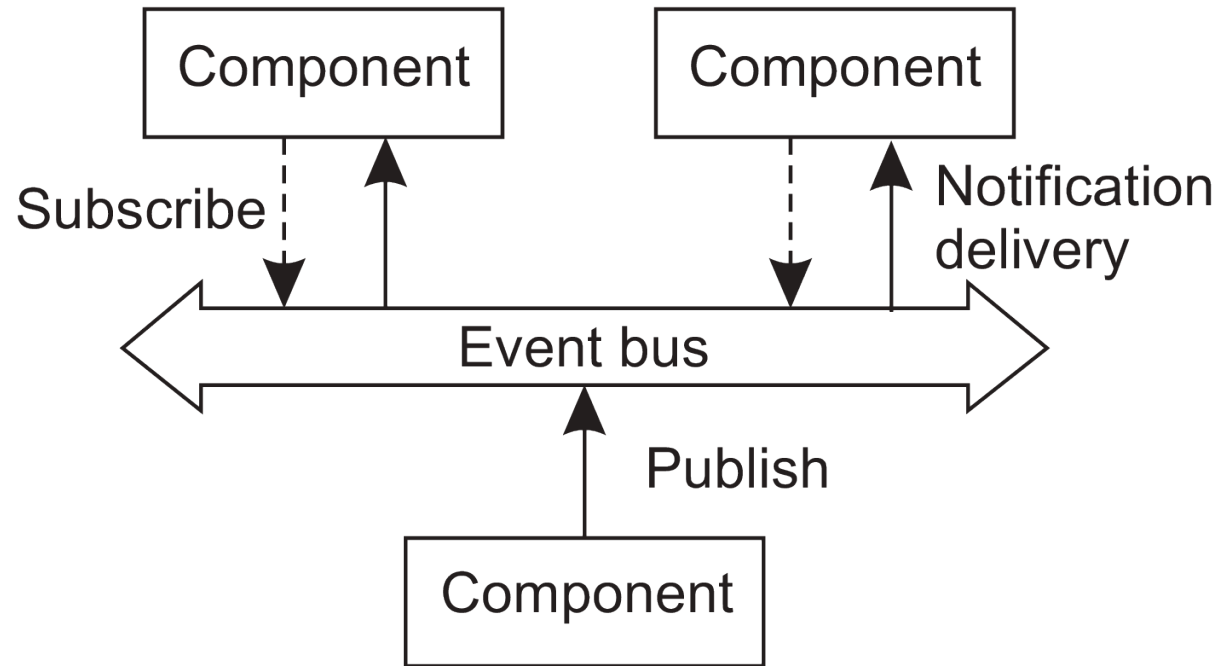


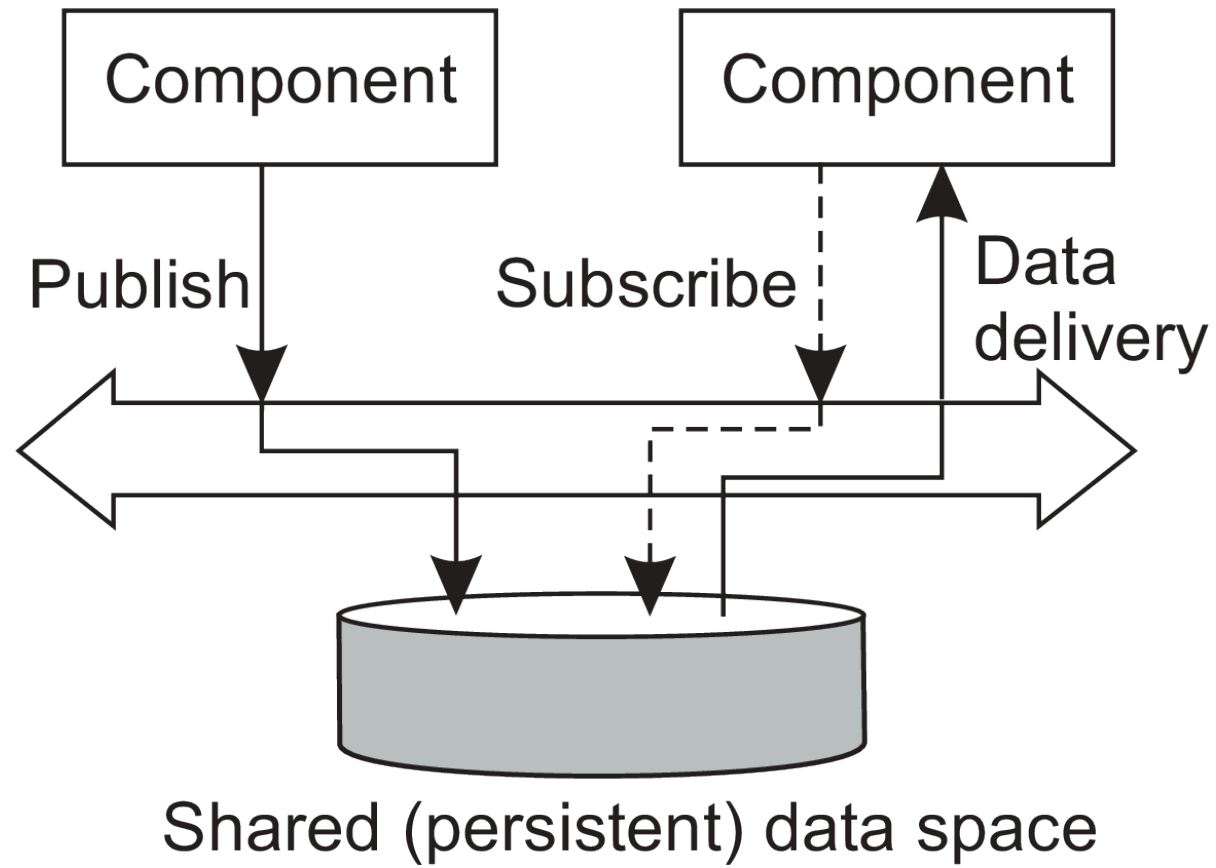
Ports and Adaptors architecture

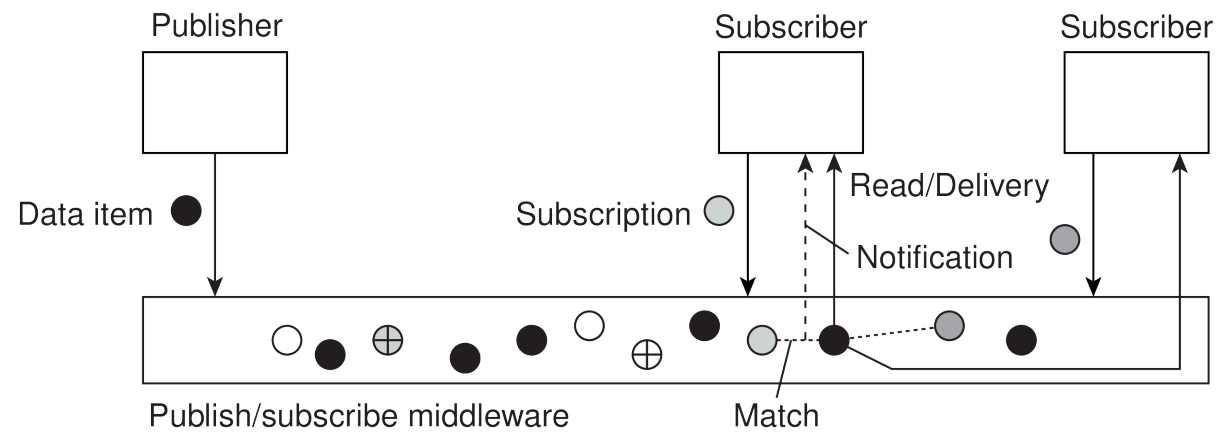


Publish-subscribe Architekturen

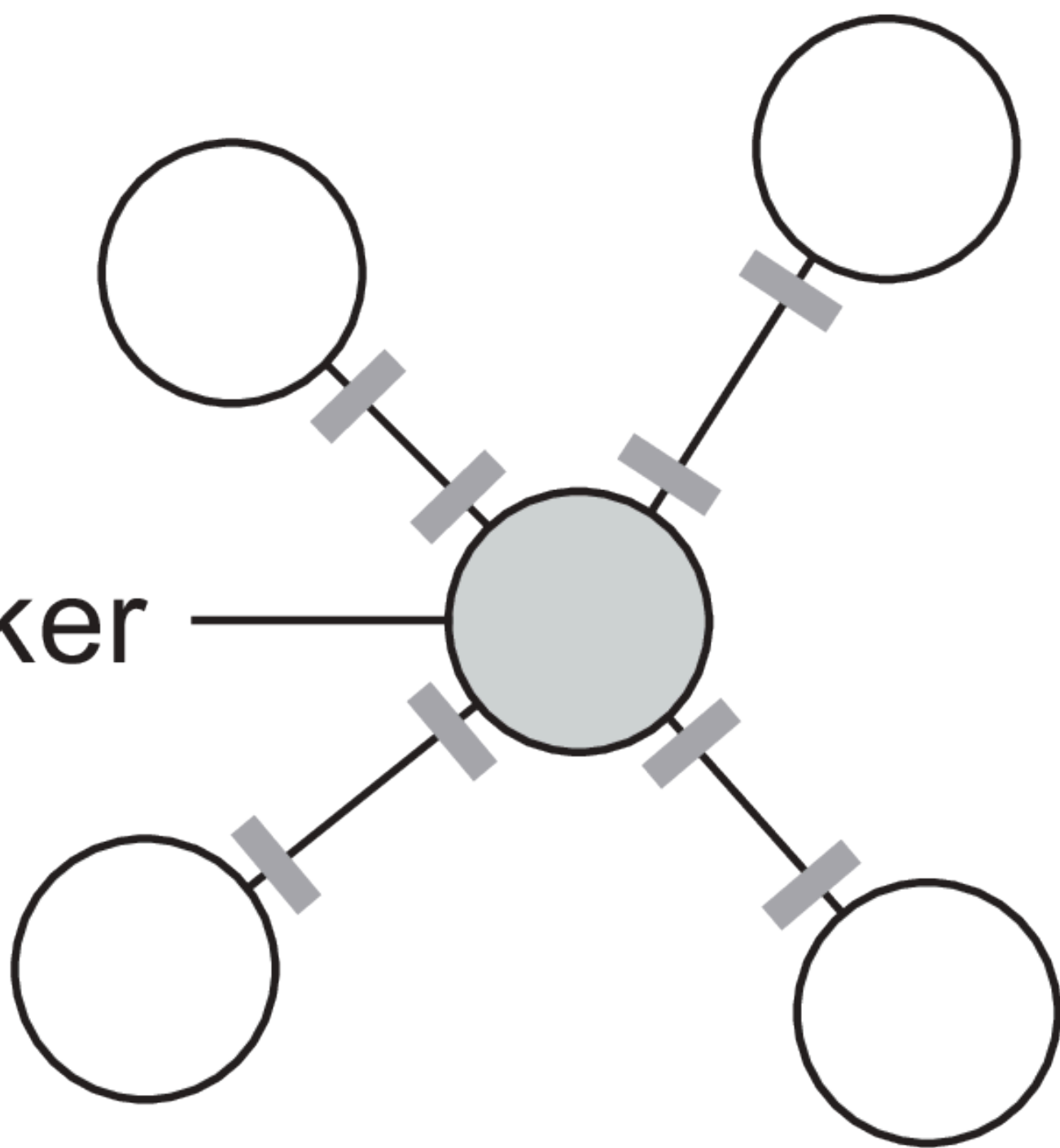
	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

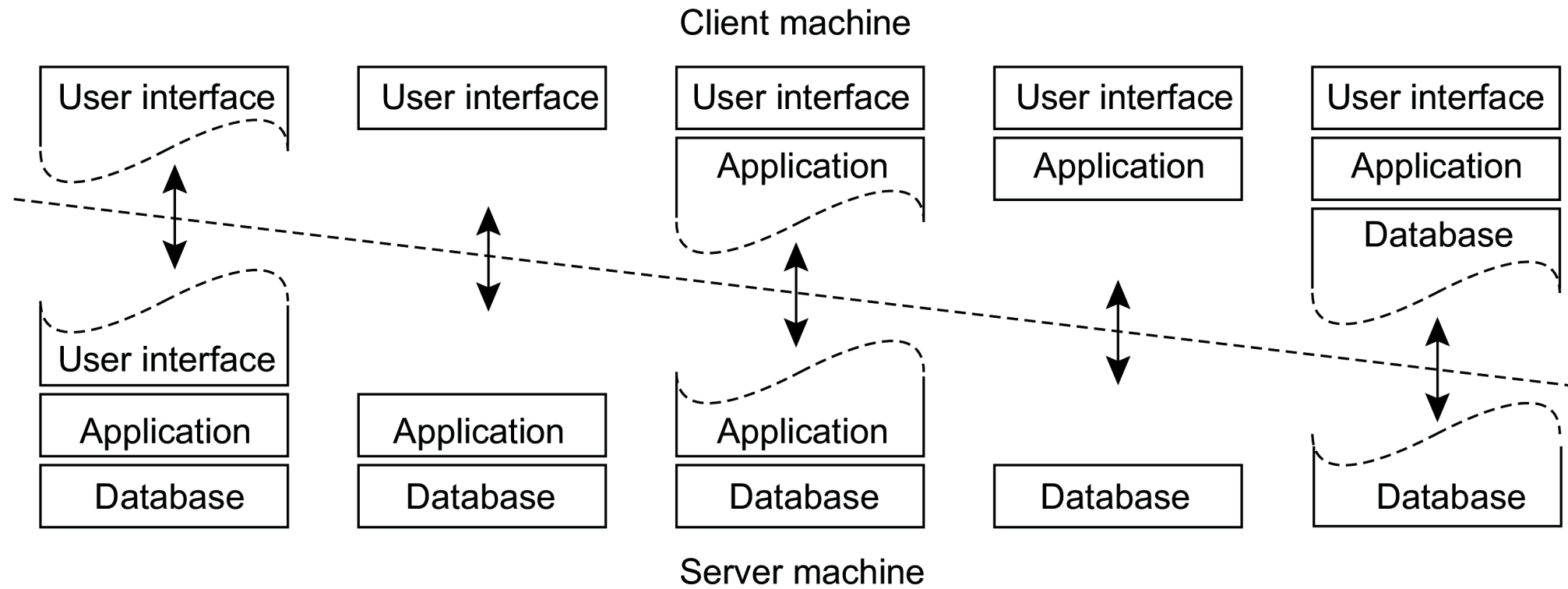






Broker



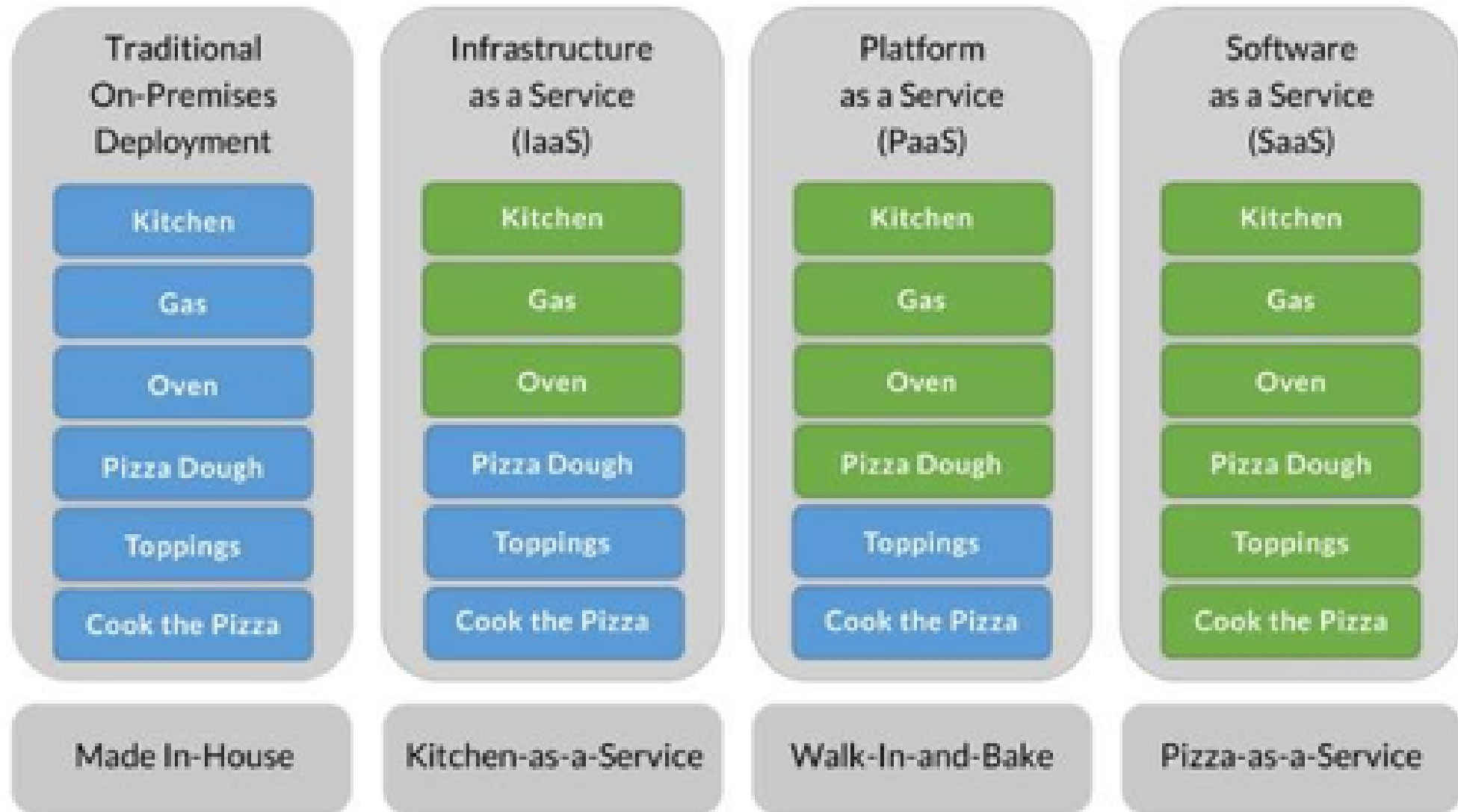


Cloud Computing

| The entire history of software engineering is that of the rise in levels of abstraction.

-- Grady Booch

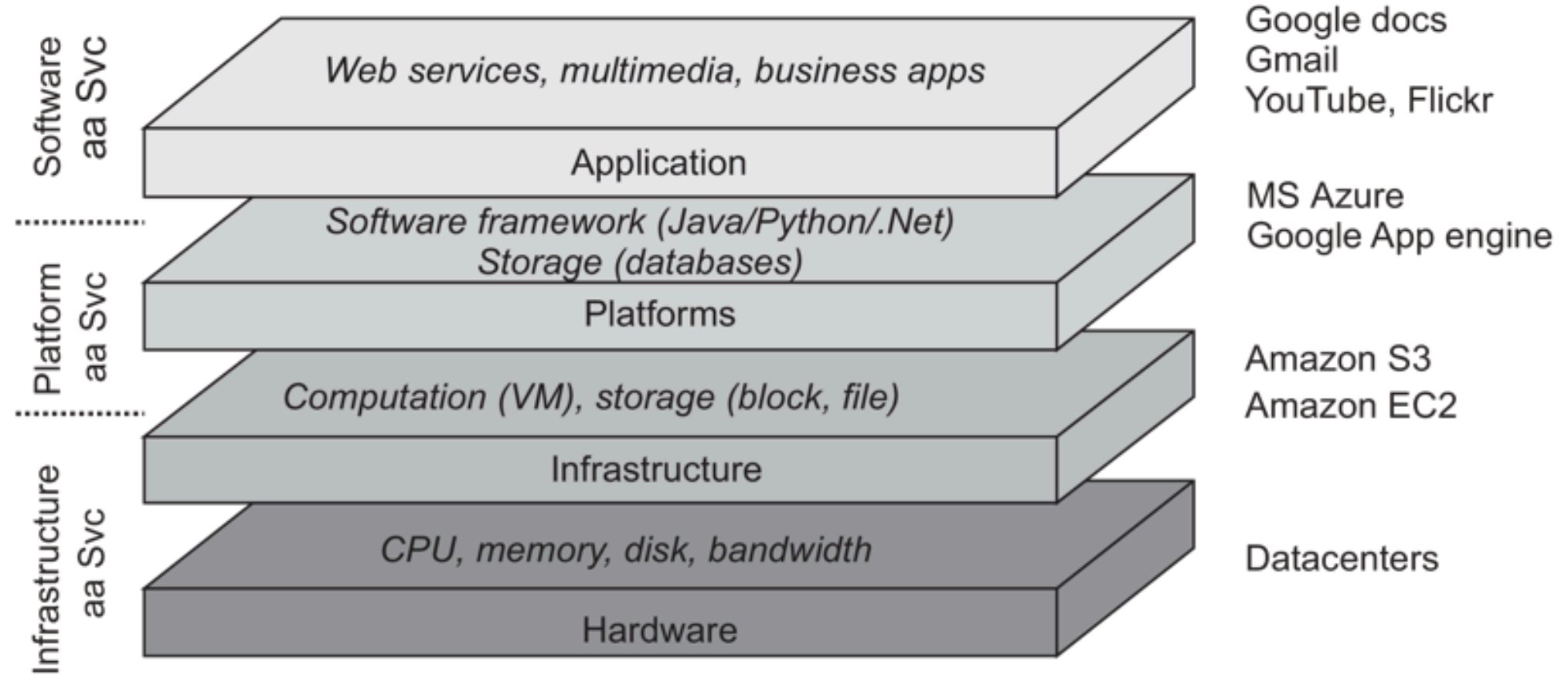
New Pizza as a Service



■ You Manage

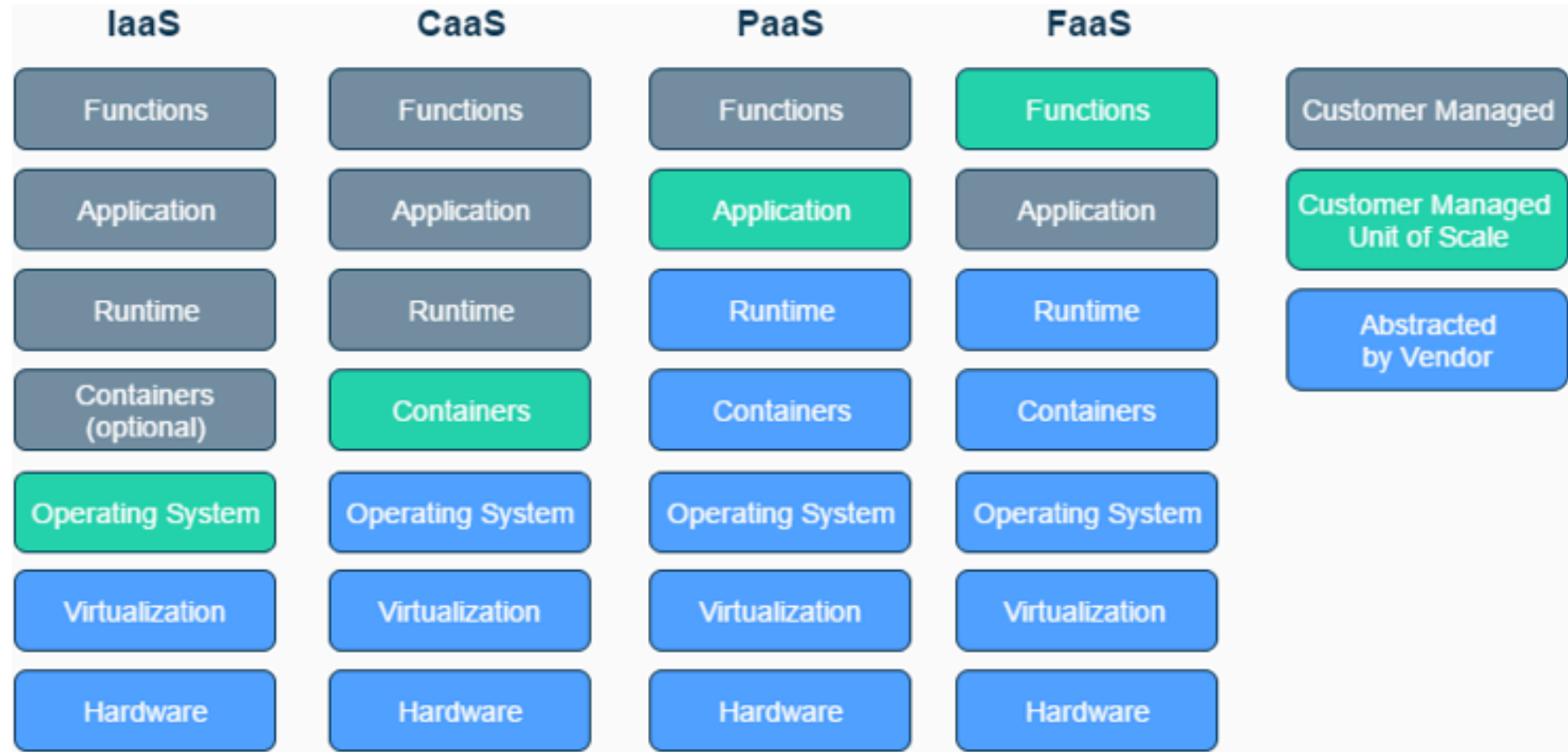
■ Vendor Manages

Abstractions

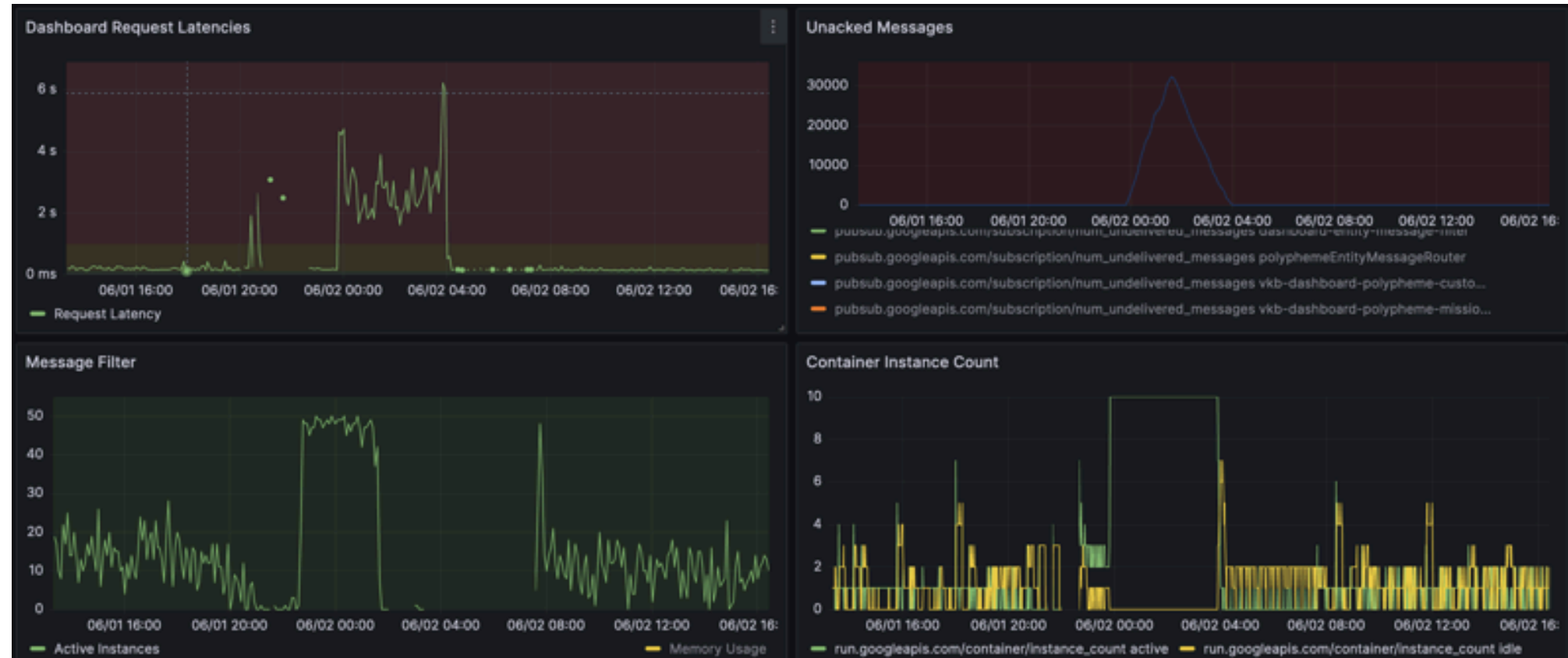


(VanSteen, 2017, S. 30)

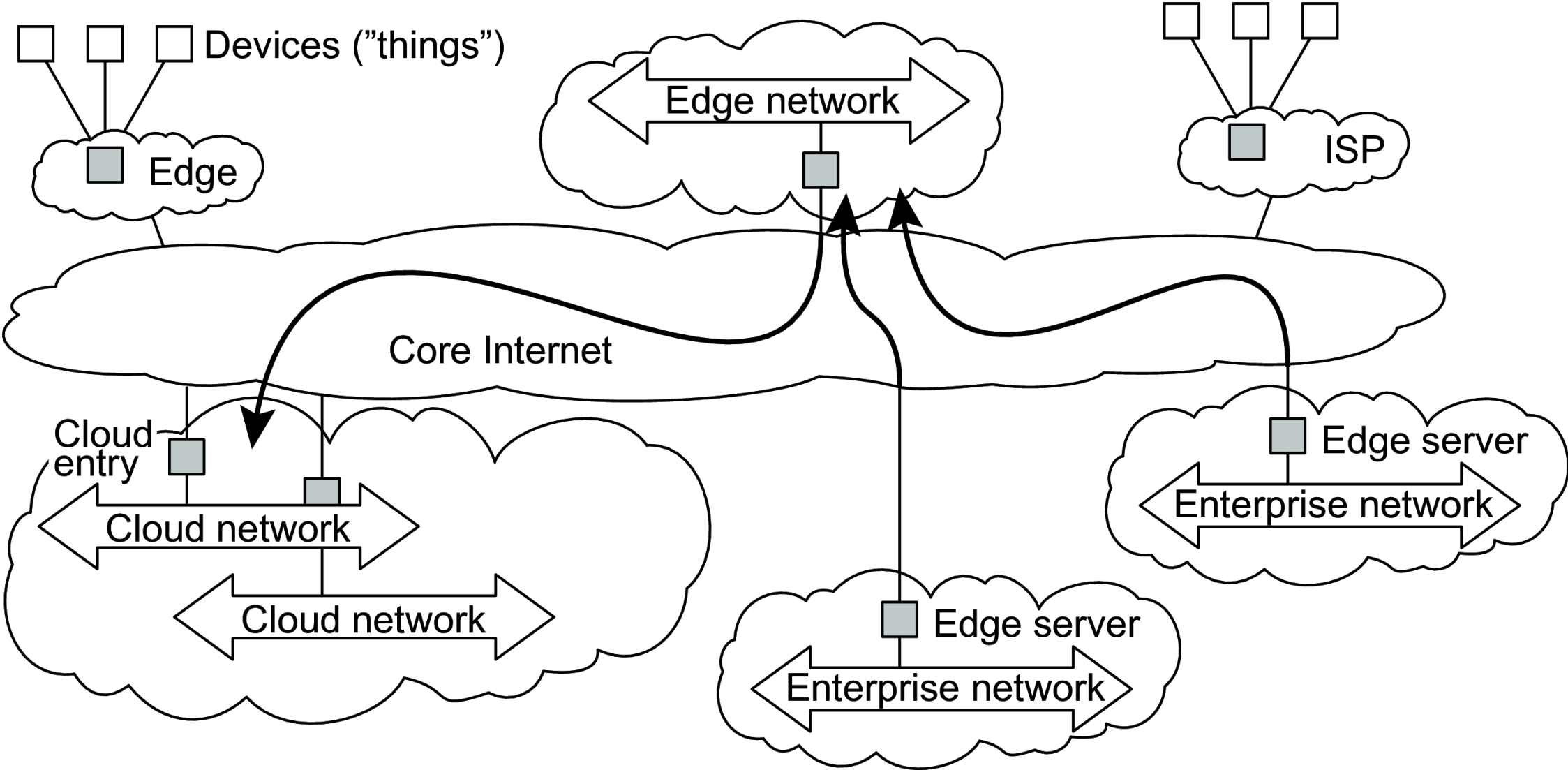
XaaS



Fallstudie



Edge Computing



Koordination

- Tasks können gleichzeitig ausgeführt werden
- Gleichzeitiger Zugriff auf gemeinsame Daten kann in inkonsistenten Daten resultieren

Mutex

- MUTual EXclusion: wechselseitiger Ausschluss
- Einfachste Möglichkeit, Ressourcen für alle anderen zu blockieren
- Critical Section wird mit acquire() und release() umschlossen
- acquire() und release() müssen atomare Operationen sein (Hardwareunterstützung)

Mutex

```
acquire() {  
    while (!available)  
        /* busy wait */  
        available = false;;  
}  
  
release() {  
    available = true;  
}
```

```
do {  
    acquire lock  
    critical section  
    release lock  
    remainder section  
} while (true);
```

Semaphore

- Mehr Möglichkeiten als Mutex
- Schützt gemeinsame Ressourcen
- Counting semaphore: Mehrere Ressourcen
- Binary semaphore: Nur eine Ressource
- Ein Zugriff auf eine gemeinsame Ressource wird mit dem Nehmen und Geben umschlossen

Beispiel

```
$semaphore = $this->createSemaphore($id);

sem_acquire($semaphore);
try {
    $entityPublishTime = $this->getEntityPublishTime($model, $id);

    if ($entityPublishTime < $messagePublishTime) {
        $returnCode = $this->saveStateToModel($model, $state, $data->timestamp);
    } else {
        $returnCode = 3;
    }
} finally {    // make sure to always release semaphore
    sem_release($semaphore);
}
```