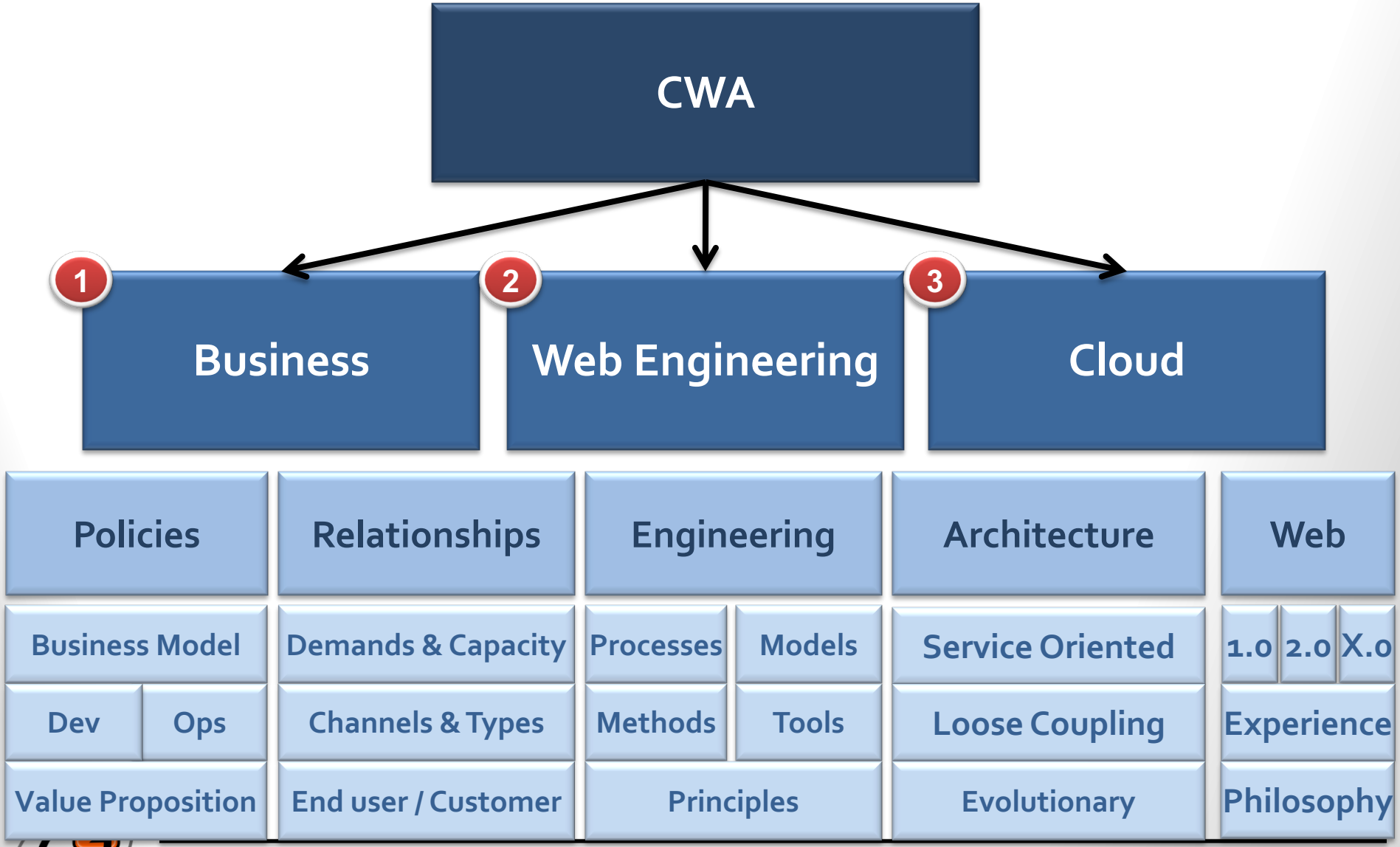
Cloud & Web Anwendungen

Prof. Dr.-Ing. Martin Gaedke
Technische Universität Chemnitz
Fakultät für Informatik
Verteilte und selbstorganisierende Rechnersysteme

Lecture Outline



PART III

- Cloud Computing

Part III – Overview

- Cloud Computing Introduction
- Cloud Computing Patterns

CHAPTER://1

- Cloud Computing Introduction

Situation

- The Future Of **Cloud Computing**
  - Oppertunities For European Cloud Computing Beyond 2010
  - Expert Group Report edited by Keith Jeffrey [ERCIM] and Burkhard Neidecker-Lutz [SAP Research]
  - Workshop results from January 2010
  - http://cordis.europa.eu/fp7/ict/ssai/events-20100126-cloud-computing_en.html

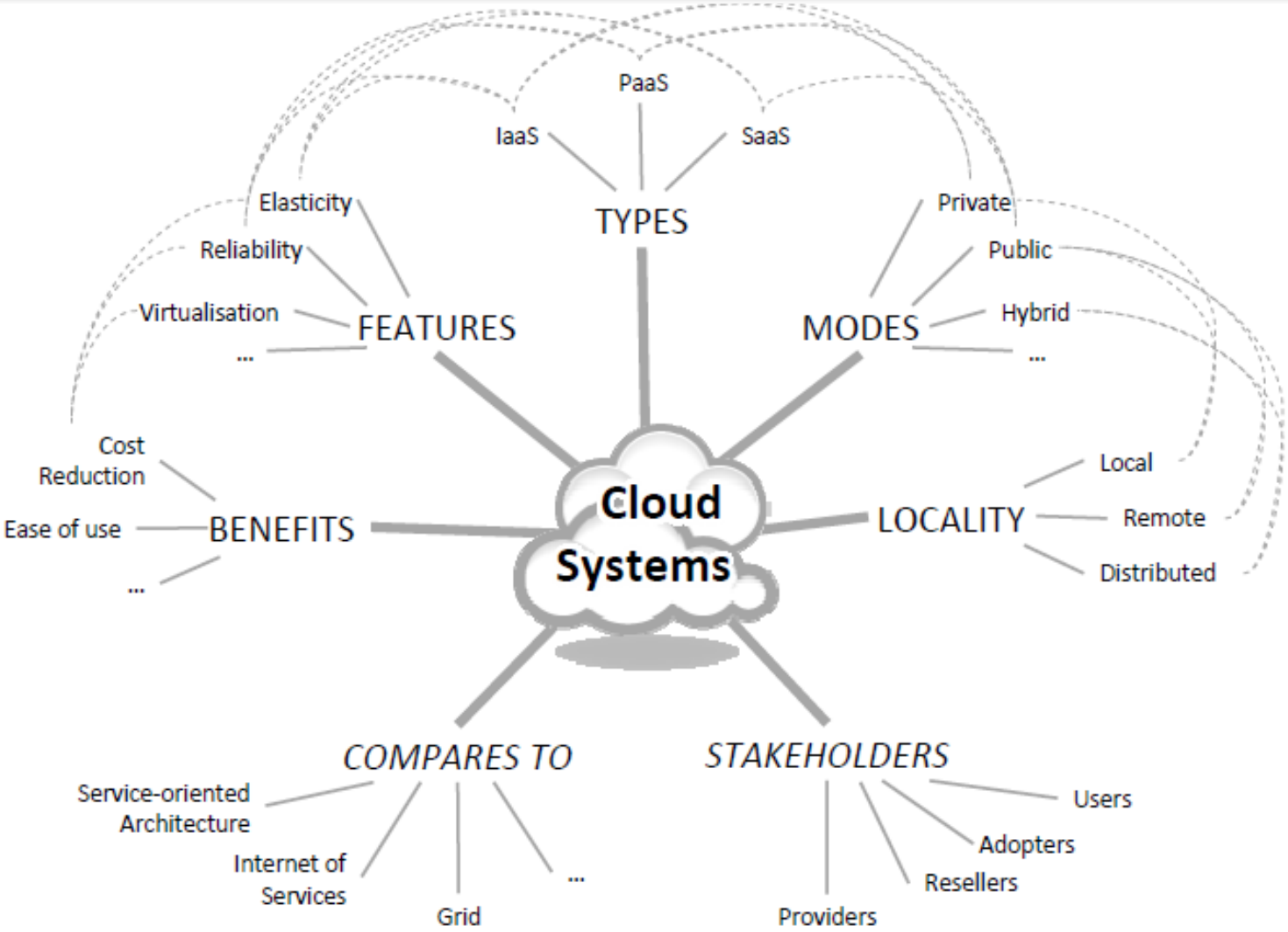
Cloud Computing Definition

cloud (klaʊd) ***an elastic execution*** *environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality (of service).*

Definition of Expert Group Report

From http://cordis.europa.eu/fp7/ict/ssai/events-20100126-cloud-computing_en.html

Non-exhaustive View



Cloud Computing

- Cloud Computing is linked intimately with those of
  - IaaS (Infrastructure as a Service)
  - SaaS (Software as a Service)
  - PaaS (Plattform as a Service)
- Collectively called *aaS (Everything as a Service)
- all of which imply a service-oriented architecture.

Cloud Computing

- vs SOA
- vs GRID
- vs Fog computing

Cloud Computing

- Cloud Systems ≠ yet another resource provisioning infrastructure
- Cloud Computing Environment
  - Infinitely scalable
  - Provides infrastructure for platforms (for apps or services)
  - Used for every purpose
  - Shifts costs for a business
  - CAPEX to OPEX (Capital to Operational Expenditures)

Current Challenges in Cloud Computing

- Current Challenges
  - Cloud offerings are heterogeneous
  - Concerns over security
  - Concerns over availability
  - Concerns over data shipping and broadband speeds

Cloud Computing

- **Microsoft OS Cloud Windows Azure Data Center - Google and Amazon battle**
  - ▶ http://www.youtube.com/watch?v=K3b5Ca6lzqE
- **The Google Data Center - Cloud Computing Storage ,Cloud Computing, File Storage**
  - ▶ http://www.youtube.com/watch?v=4SkVT-nNgao
- **Streetview inside Google Data Center Lenoir, NC**
  - ▶ http://www.google.com/about/datacenters/inside/streetview/

CHAPTER://2

- Cloud Computing Patterns

Adapted from
Wilder, B. (2012). *Cloud Architecture Patterns*.
(R. Roumeliotis, Ed.) (1st ed.). Sebastopol, CA, USA: O'Reilly Media.

SECTION://1

- Introduction

Cloud Platform Characteristics

- Enabled by „infinite" resources, limited by maximum capacity of individual VMs, **cloud scaling is horizontal**
- Enabled by short-term resource rental model, **cloud scaling releases resources as easily as they are added**
- Enabled by a metered pay-for-use model, **cloud applications only pay for currently allocated resources**
- Enabled by self-service, on-demand, programmatic provisioning/releasing of resources, **cloud scaling is automatable**
- Enabled & Constrained by multitenant services, cloud applications are optimized for cost rather than reliability: **failure is routine, but downtime is rare**
- Enabled by a **rich ecosystem of managed platform services** such as for VMs, data storage, messaging and networking, cloud application development is simplified

Cloud-Native Application

- **Leverages cloud-platform services** for reliable, scalable infrastructure („*Let the platform do the hard stuff*.“)
- Uses **non-blocking asynchronous communication** in a **loosely coupled architecture**
- **Scales horizontally**, adding resources as demand increases and releasing resources as demand decreases
- Cost-optimizes to run efficiently, not wasting resources
- **Scales automatically** using proactive and reactive actions
- Handles scaling events & transient failures without user experience degradation
- Handles node & transient failures without downtime

SECTION://2

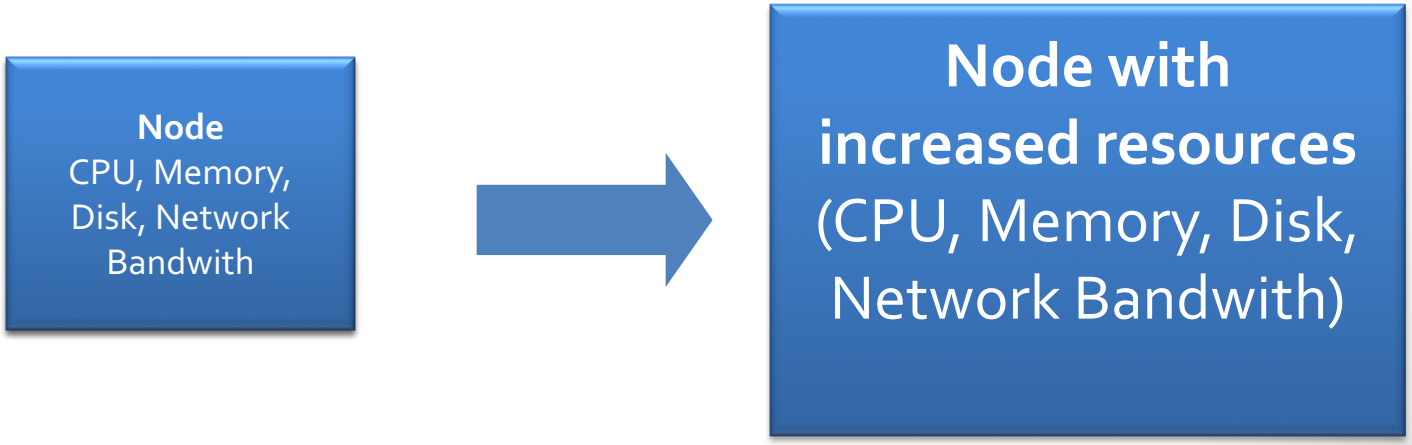
- Horizontally Scaling Compute Pattern

Scalibility

- Consider
  - Concurrent users
  - Response time
- Scale Unit
  - e.g. for every 100 users: 2 web server nodes, 1 service node, 100MB disk space
- Performance vs. Scalibility
  - Performance: experience of a *single user*
  - Scalibility: *number of users* who still have a positive experience

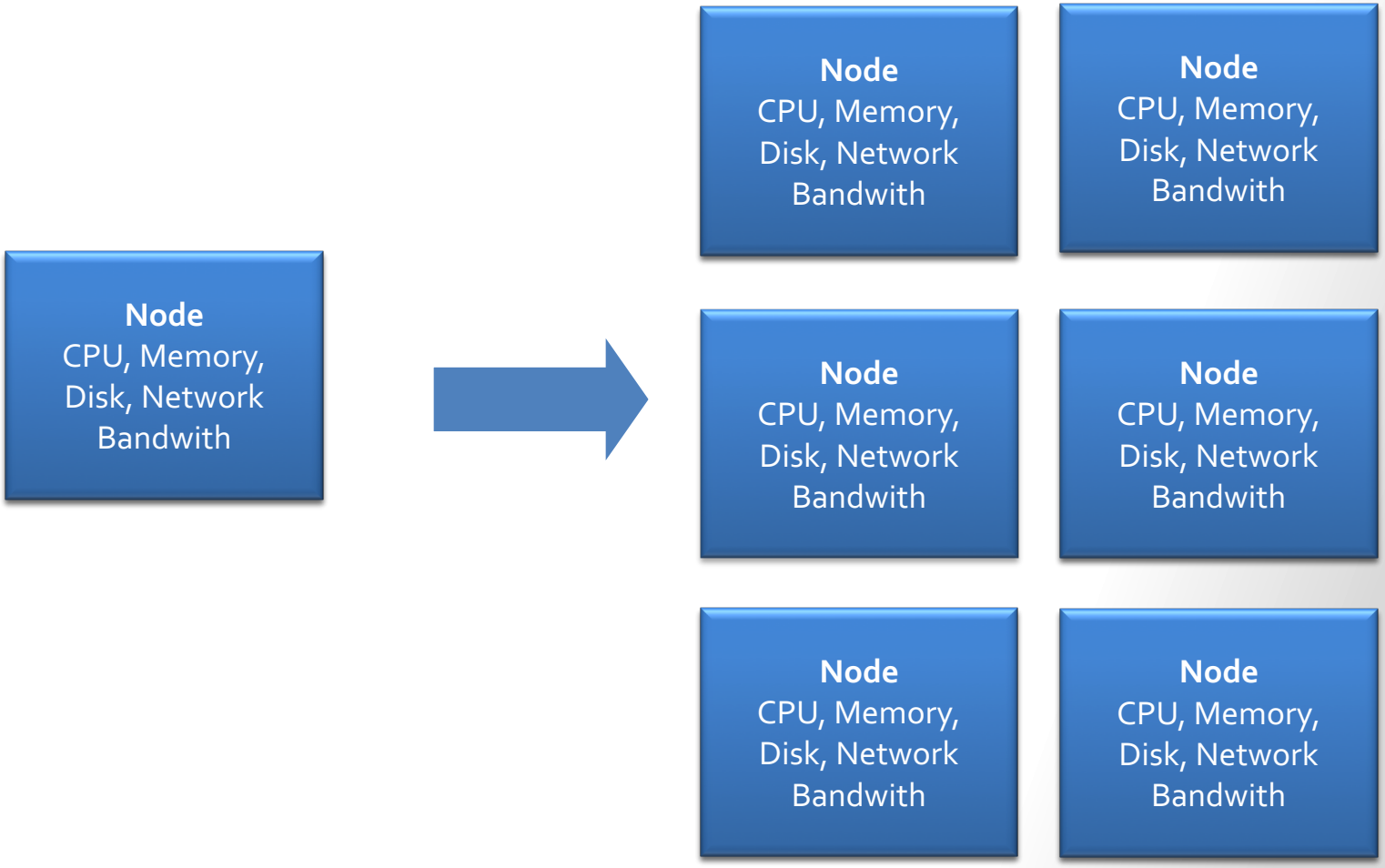
Vertical vs. Horizontal

- Scale up



Vertical vs. Horizontal

- Scale out



When to apply Horizontal Scaling

- Cost-efficient scaling of compute nodes is required
- Application capacity requirements exceed the capacity of the largest available compute node
- Application capacity requirements vary seasonally, monthly etc., or are subject to unpreictable spikes in usage
- Application compute nodes require minimal downtime

Horizontally Scaling Compute Pattern

- Cloud Scaling is **reversible**
- **Elasticity**
  - Add additional compute nodes when required
  - Release compute nodes when no longer required
- **Cost-oriented**
  - Down-scaling saves money
- **Virtualization** as enabling technology
  - Horizontal scaling is achieved adding/removing Virtual Machines with identical configuration (image) as compute nodes

Session state without stateful nodes

- For efficient horizontal scaling, session state cannot be stored locally on the nodes
- Session state has to be stored externally
  - use Cookies & LocalStorage if size allows
  - else use NoSQL data stores, cloud storage, distribute caches, store session identifiers in Cookies
- Allows individual nodes to stay autonomous
- Scalability issue is shifted to the storage mechanism

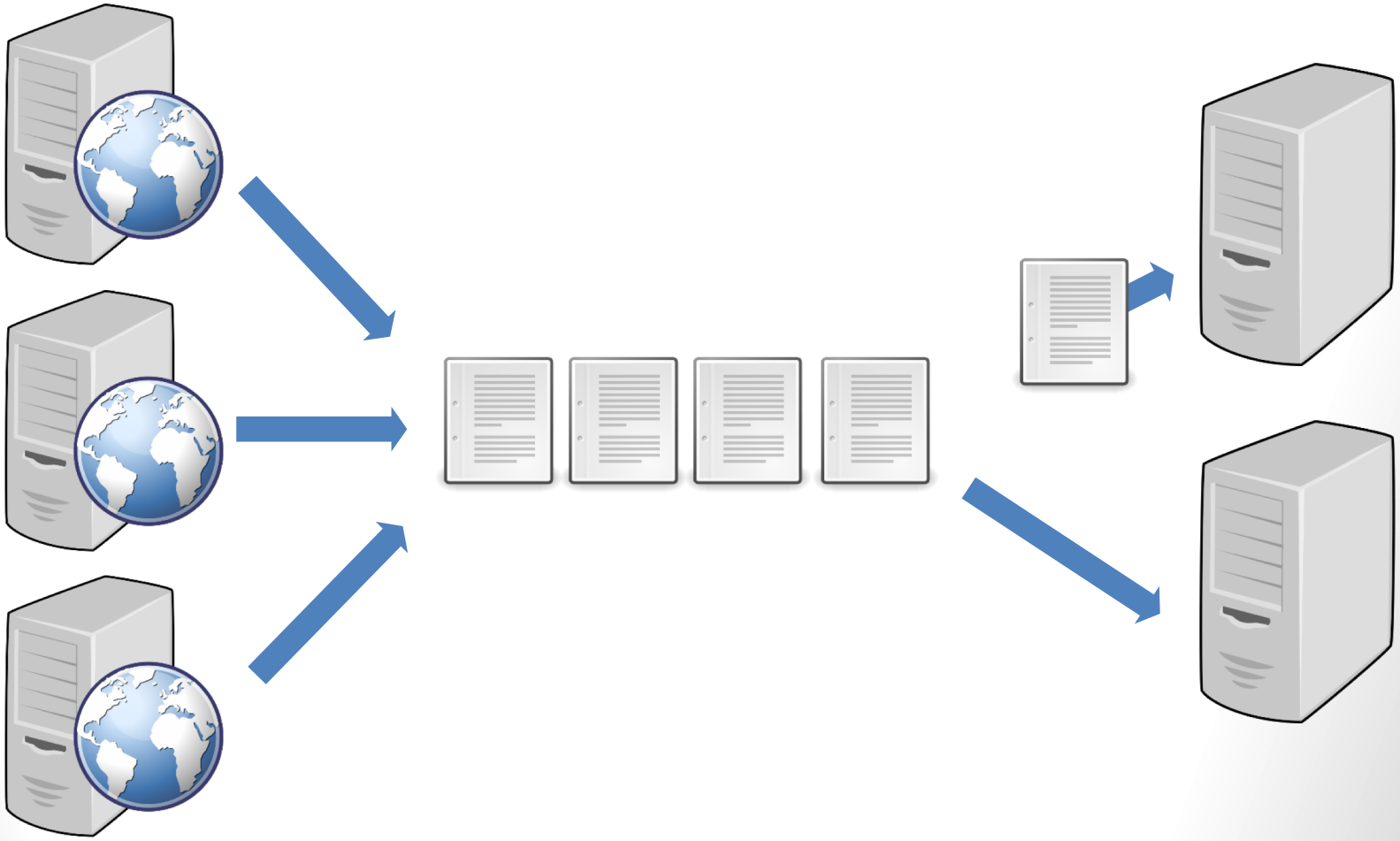
SECTION://2

- Queue-Centric Workflow Pattern

When to apply Queue-Centric Workflow

- Application is decoupled across tiers, though tiers still need to collaborate
- Application needs to guarantee *at-least-once processing* of messages across tiers
- A consistently responsive user experience is expected in the user interface tier, even though
  - dependent processing happens in other tiers
  - third-party services are accessed during processing

Queue-Centric Workflow Pattern



Queue-Centric Workflow Pattern

- UI (Web) Tier adds messages to the queue
- Requires *reliable queue* (usually as service provided by cloud platform)
- Receiver (at-least-once processing)
  1. Get next available message from the queue and mark as hidden
  2. Process message
  3. Delete message from queue
- Invisibility Window
  - ► Message is hidden for limited period of time
  - ► Automatic reappearance of messages if processing time exceeds invisibility window and no refresh → at-least-once

Queue-Centric Workflow Pattern

- Dequeue count
  - Increased for each processing attempt of a message
- Additional step indicators for multi-step processes, if failure continue from last completed step
- Poison messages
  - Erroneous messages, cannot be processed successfully
  - Detect: dequeue count > N, depending on processing time
  - Handle: remove from queue, important messages: add to dead letter queue for later inspection
- Beware of possible money leak due to queue service
  - Queue service calls (e.g. dequeue) cause costs
  - Avoid fast polling

SECTION://3

- Auto-Scaling Pattern

When to apply Auto-Scaling

- Cost-efficient scaling of computational resources
- Continous monitoring of fluctuating resources is needed to maximize cost savings and avoid delays
- Frequent scaling requirements involve cloud resources such as compute nodes, data storage, queues, or other elastic components

Auto-Scaling Pattern

- Schedule for known **events** (e.g. Champoins league, business hours)
- Create **rules** to react to environmental **signals**
  - Fixed dates/times
  - Average response time & Queue length
  - Memory & CPU utilization
  - Node failures
- Rules
  - Increase/Decrease nodes (cf. scale units)
  - *N+1 rule* deploy N+1 nodes if N nodes are required to buffer sudden spikes & cope with node failure
- Use upper/lower bounds to constrain auto-scaling rules

Example: WASABi rules

```
<rules xmlns="http://schemas.microsoft.com/practices/2011/entlib/autoscaling/rules">
  <constraintRules>
    <rule name="peek" enabled="true" rank="100">
      <timetable duration="10:00:00" startTime="08:00:00" utcOffset="2">
        <weekly days="Sunday"/>
      </timetable>
      <actions><range target="WebRole1" min="20" max="40"/></actions>
    </rule>
  </constraintRules>
  <reactiveRules>
    <rule name="notbusy" enabled="true" rank="1">
      <when><lessOrEqual operand="cpu" than="20"/></when>
      <actions><scale by="-1" target="webrole1"/></actions>
    </rule>
    <rule name="queueIsFull" enabled="true" rank="1">
      <when><greaterOrEqual operand="queue" than="10"/></when>
      <actions><scale by="1" target="webrole1"/></actions>
    </rule>
  </reactiveRules>
  <operands>
    <performanceCounter alias="cpu" performanceCounterName="\Proccessor(_Total\% Processor Time)"
                        source="WebRole1" aggregate="Average" timespan="00:15:00" />
    <queueLength alias="queue" queue="queue1" aggregate="Average" timespan="00:10:00"/>
  </operands>
</rules>
```

Adapted from: http://blogs.microsoft.co.il/blogs/applisec/archive/2012/01/11/azure-elasticity-with-wasabi-application-block.aspx

SECTION://4

- Map Reduce Pattern

Eventual Consistency

- CAP Theorem: A distributed system cannot satisfy all of the following three guarantees
  - Consistency (all queries yield the same data at the same time)
  - Availability (all requests served either successfully or with failure notification)
  - Partition tolerance (correct operation, even if some nodes fail)
- Eventual Consistency
  - trades Consistency for Availability and Partition tolerance

Eventual Consistency

- When updates occur, data is not *immediately consistent* across all nodes
- Data will be consistent after a delay
- *Stale data* is data which is no longer current
- Example: DNS
  - Propagating updates can take several hours
- ACID vs. BASE
  - Atomicity, Consistency, Isolation, Durability
  - Basically Available, Soft State, Eventually Consistent

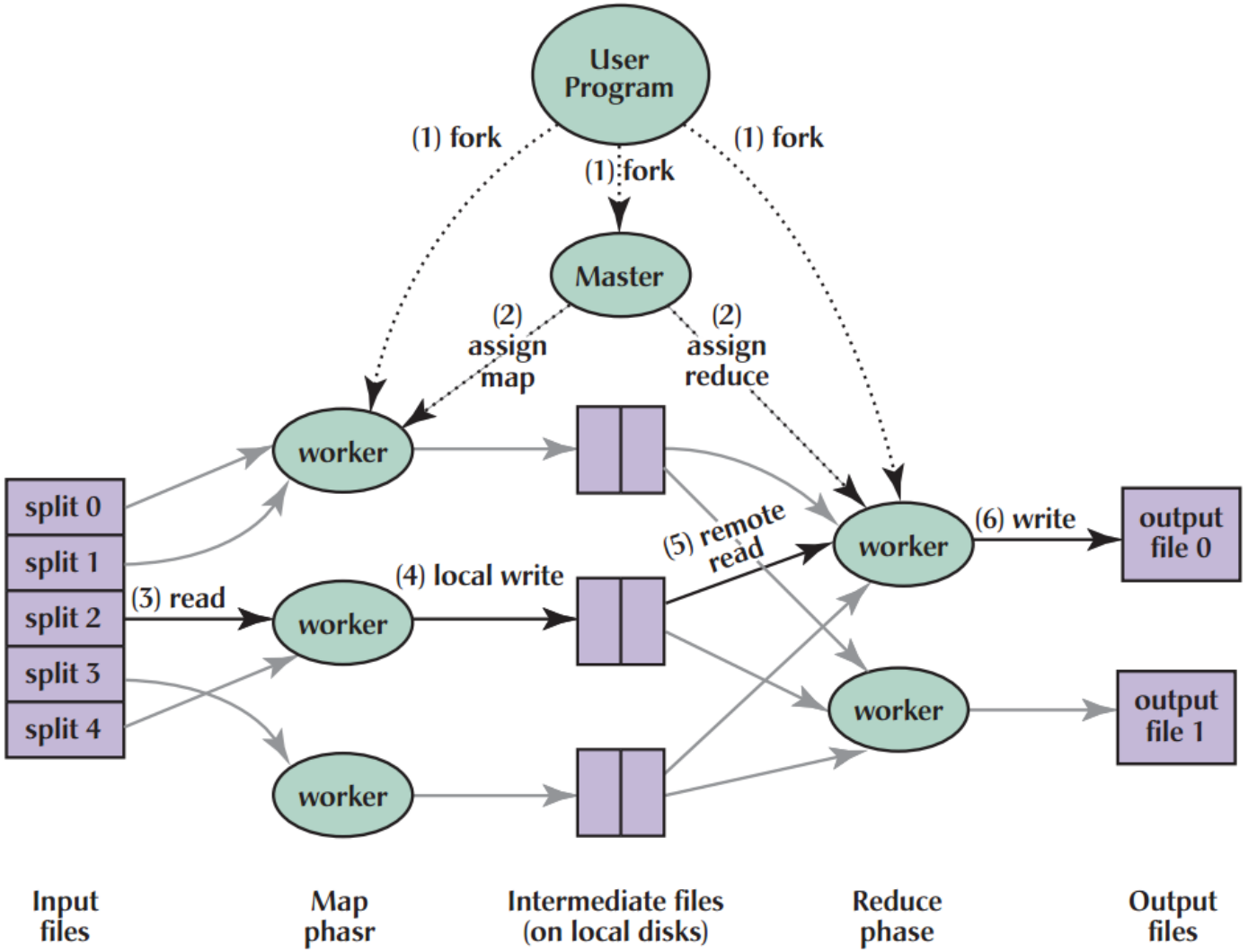
When to apply MapReduce

- Application processes large volumes of data (structured, semi-structured, unstructured) stored in the cloud
- Data analysis requirements change frequently or are ad hoc
- Application requires reports beyond capabilities of traditional database reporting tools because input data is too large or no in compatible structure

MapReduce

- Simple programming model for processing highly **parallelizable** data sets
- Batch processing approach
- Two phases: Map & Reduce
  - **Map** function applied to each input element (key-value pairs) produces intermediate results (key-value pairs)
  - Wait for all Map functions to complete, apply **Reduce** function to each intermediate result producing final results
- All Map functions are independent → can be run in parallel
- All Reduce functions are independent → can be run in parallel
- MapReduce as Service offered by Cloud Providers
  - Hadoop (as a Service)

MapReduce Execution



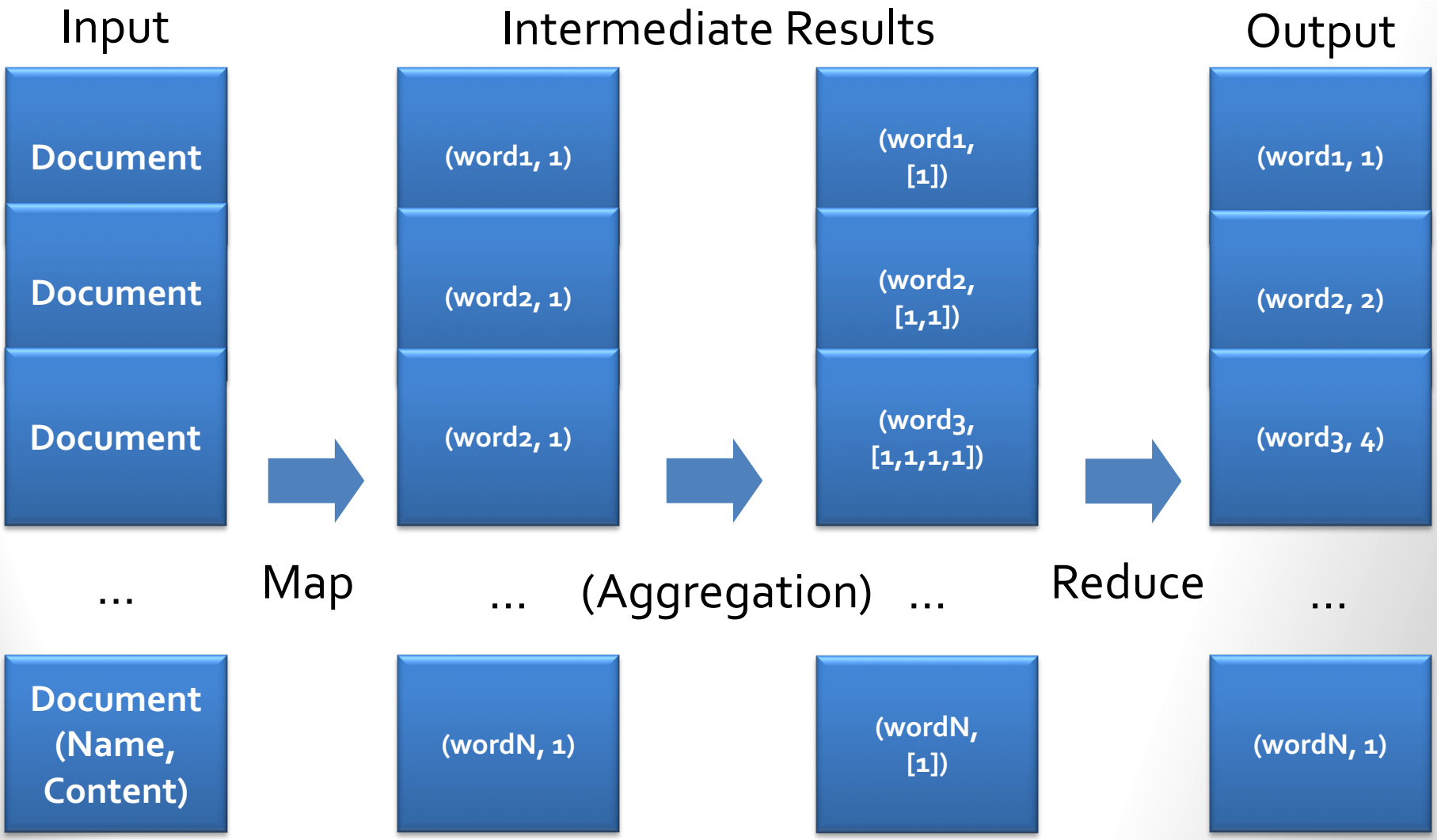
From: Dean, J., & Ghemawat, S. (2008). MapReduce. *Communications of the ACM*, *51*(1), 107–113.

MapReduce Example Word Count

```
def map(documentName, documentContent):
    for line in documentContent:
        words = line.split(" ")
        for word in words:
            EmitIntermediate(word, 1)

def reduce(word, counts):
    wordCount = 0
    for count in counts:
        wordCount += count
    Emit(word, wordCount)
```

MapReduce Example Word Count



MapReduce Example Word Count in Pig

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS
(line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

Source: http://en.wikipedia.org/wiki/Pig_(programming_tool)