Aalto University
School of Electrical
Engineering

Network communication

25.1.2019

Contents

- **Communication**
  - Characteristics of physical and protocol layers
  - Models, protocols and coordination
- **Communication in microservice architectures**
  - Failures
- **Coupling and modularity**
  - Types of coupling in different architectural approaches
  - Achieving modularity at different quanta

Communication networks

- **Even "virtualized" networks operate in physical reality**
  - Sometimes can assume locality e.g. loopback speeds (pods)
  - Distribution and decentralization may hide physical aspects
    - *Translation: System might be placed to straddle a buggy or oversubscribed router/switch*
  - Physical latencies: speed of light & electric signals, processing delays in switches and routers; retransmits

~ 40 000 km
* ½
/ 300 000 km/s
/ 2/3
= 100 ms

+ amplification delay
+ routing delay

Communication networks

- **Assumption of “infinite network capacity” in cloud may fail**
  - Loss of 50% of network capacity in a datacenter (backhoe)
  - Limits at virtual machines and physical servers (AWS ENA 25 Gbps)

  → even if you cannot saturate an IaaS PoP, you can closer to your service

- **Further delays and failures from protocols and OS**
  - Number of concurrent TCP connections (OS)
  - Bugs in protocol implementations (usually non-OS)

Network protocols

- **TCP almost universal, but**
  - In some situations UDP may be more suitable (within a service)
  - SCTP tends to keep popping up (alpha in K8S v1.12)
- **IPv6**
  - Getting more common, but still mostly user-side requirement
- **Deep magic**
  - TCP slow start algorithm — originally for congestion control
  - Anycast, multicast and broadcast (if you control subnets and/or routers)
  - VPNs and tunnels sometimes for integration (island hopping)

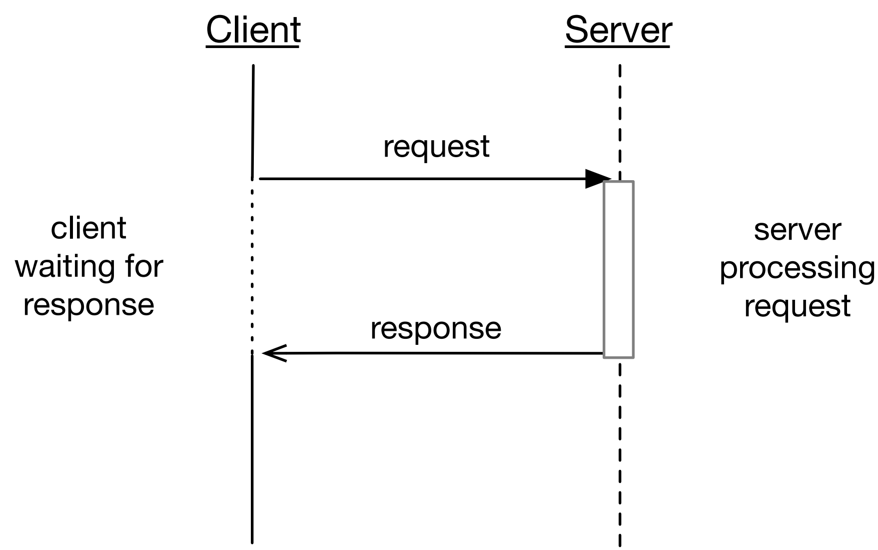
Application protocols

- **Almost all service interactions occur at application level protocols**
  - HTTP and HTTPS primary (QUIC in the future?)
    - *HTTP(S) used to transport other application level protocols*
    - *SOAP, REST, …*
  - gRPC, Thrift, AQMP, etc.
- **Operate on top of TCP**
  - Sometime work around TCP issues (such as slow start, with Keep-Alive connections)
  - TCP is connection-oriented: connect → transmit → close
  - Usually client-server, e.g. specific listener <u>address</u> and <u>port</u>

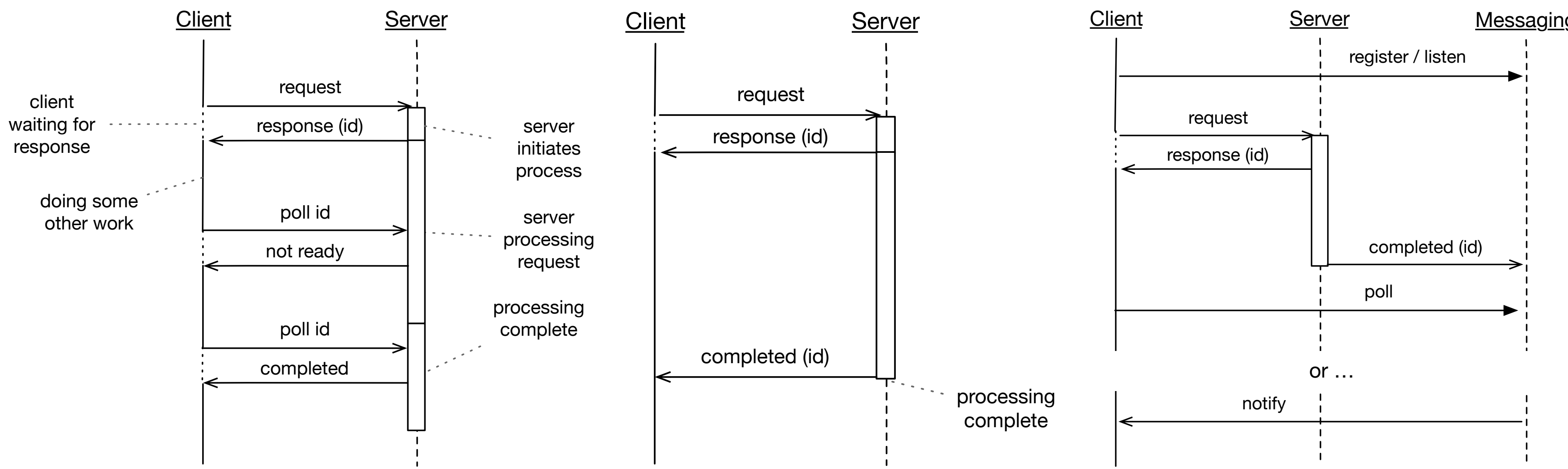
Communication models

- **Synchronous response**
  - Request-response pattern
  - Reply expected immediately (after processing)
- **Asynchronous response**
  - Processing started by request
  - Immediate response provides a handle or identifier
  - Response methods
    - *Polling by client (known endpoint or part of response)*
    - *Callback from server (agreed-upon endpoint or part of request)*
    - *Response publish (message queue, pubsub, blackboard, ...)*
- **Message-passing**
  - Request itself asynchronous

Synchronous request



Asynchronous communication models



Asynchronous requests vs. asynchronous applications

- **Asynchronous communication is between two parties**
- **Asynchronous applications are self-contained**
  - Avoid blocking at thread level: some other method of waiting
  - Event loops, I/O selectors, continuations, etc. low-level mechanisms
  - Erlang, AKKA, asyncio programming language level constructs

- **A synchronous application can make asynchronous requests**
- **An asynchronous application can make synchronous requests**

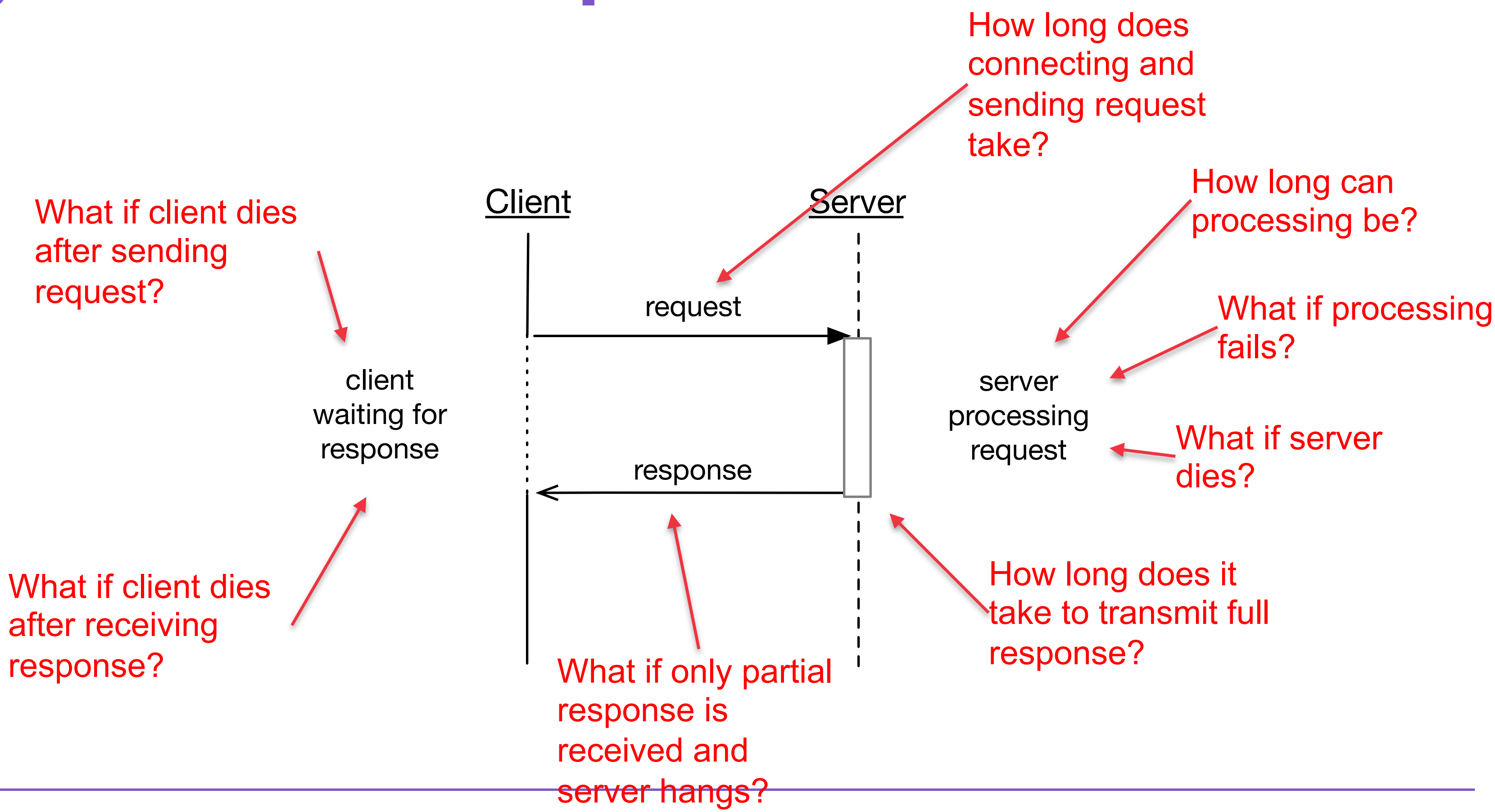
Failures

Failures in distributed systems

- **Rule of thumb:**
  - Everything fails all the time (randomly, when least expected)
- **See *Network is reliable* paper (hint: it is not)**

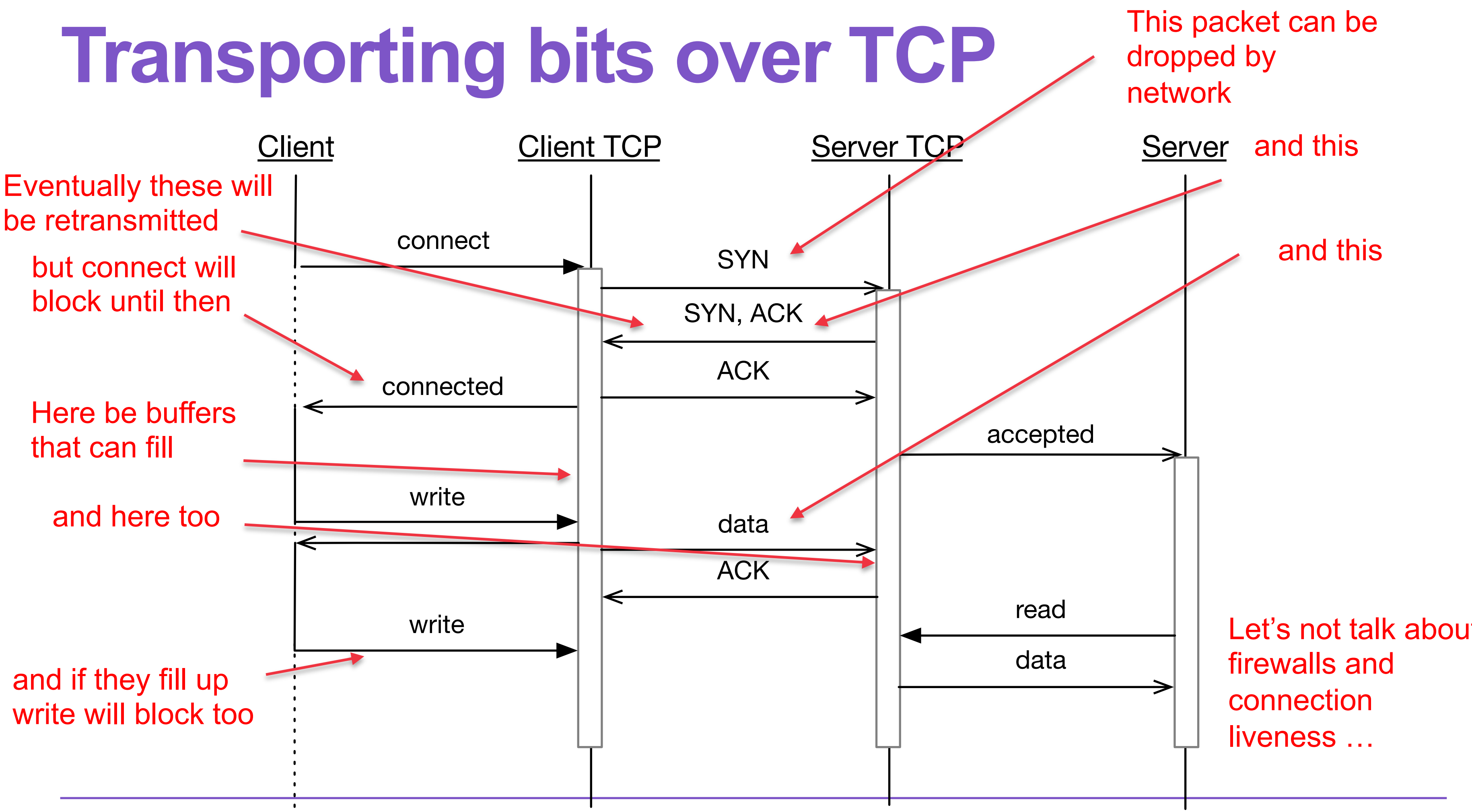
- **Microservice architectures fail more**
  - More components, more computers, more connections, more changes, more of everything
  - Risks of correlated failures can be either higher or lower than for monolithic systems
  - See first lecture slide how number of components affects reliability

Synchronous request



But wait, it gets worse!

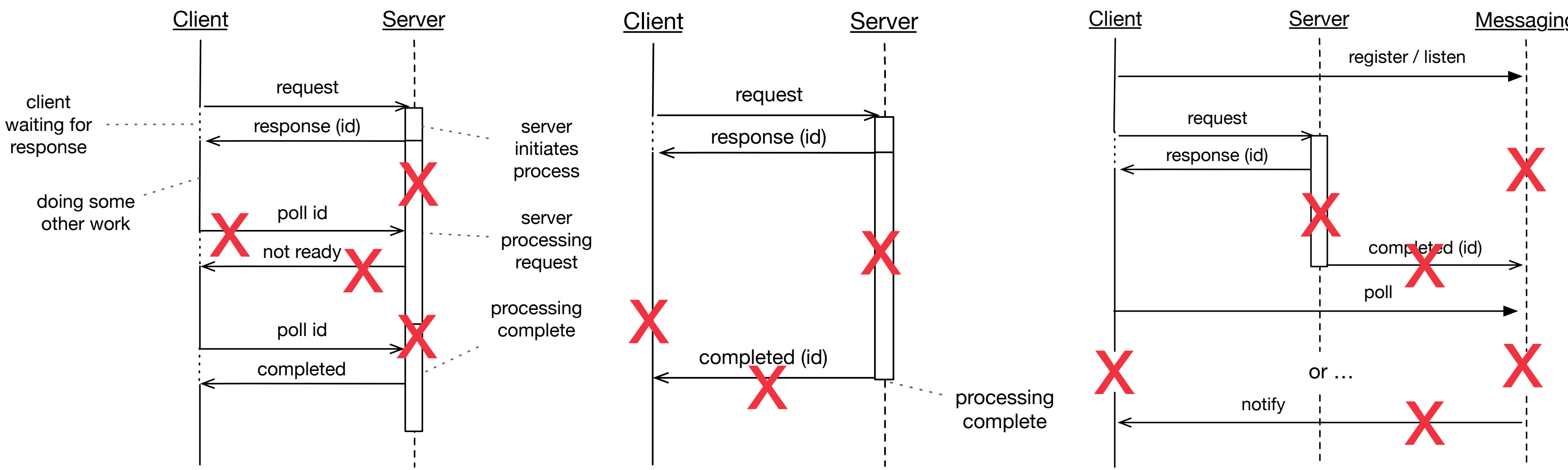
Transporting bits over TCP



You are not expected to understand why these failures occur.

Just to understand that they lurk everywhere.

Asynchronous communication models



Addressing network failures

- **Failing terribly is better than hanging indefinitely**
  - At least you can see them in logs / monitoring
- **All low-level socket operations can fail**
  - This includes close() … in Java, even it can cause an exception
- **All network operations should have timeouts**
  - Abstractions may try to hide the network (Java RMI)
- **Long-lived quiescent connections are subject to random network dropouts**
  - Stateful firewalls
  - TCP and protocol-level keepalives (ping, echo, …)

Addressing protocol failures

- **Specification:**
  - GET /resource
  - 200 OK with application/json or 503 Service Unavailable
- **Code:**
  - ```
    resp = conn.get("/resource")
    if resp.code == 200:
       j = json.loads(resp.body)
       …
    elif resp.code == 503:
        …
    ```
- **Transparent proxy**
  - May return 504 Gateway timeout
  - Might randomly respond with text/html advert page
- **We'll cover architectural approaches to handling network and remote failures later in the course**
    ```

Brewer's theorem (aka CAP)

- The CAP theorem (later proven) states that for distributed systems, out of

  - Consistency
  - Availability
  - Partition-tolerance

  it is possible to achieve only CP or AP all the time

- See this for later description of the theorem (with critique)

Brewer's theorem's consequences

- **Hard partitions are generally rare**
  - Most of the time it is possible to achieve both consistency and availability
- **However, partitions do still occur**
  - Then you need to choose between availability and consistency
  - "Eventually consistent" mechanisms choose availability

- **In large enough systems, something fails all the time**
- **Consideration in services — which is critical?**

Brewer's theorem's consequences

- Just accept that

  all the time

  it is not possible to get ACID guarantees in a distributed system

  &

  all microservice architectures are distributed systems.