Models

MDS E2015
Søren Debois
CDK Chapter 2

Meta

Change of schedule

- Mandatory exercise set 1: Next week.
  Deadline Fri, Sep 25.

- Mini-project 2: Week after.
  Deadline Fri, Oct 9.

- Please fill out MP1 survey on learnit!

Recap

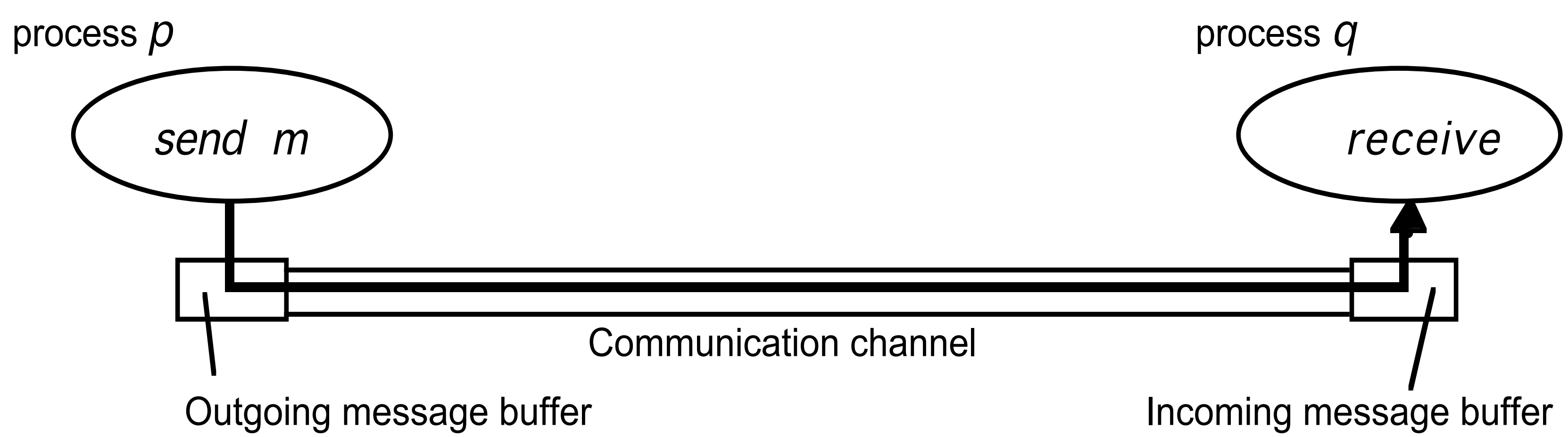
Distributed Systems

- “[A distributed system is] one in which components located at networked computers communicate and coordinate their actions only by passing messages.”

- Challenges: Heterogeneity, Openness, **Security**, Scalability, Availability, **Failure handling**, **Concurrency**, Transparency, Quality of Service.

Foundations of Networking

- Terminology: Processes, channels, failures.



- Protocol layers, each protocol adding functionality
- Protocol layer headers.

Networking

- Different kinds of network, different characteristics
- Network protocols implemented in layers, providing different guarantees and abstraction
- IPv4 -> v6: unexpected growth and mobility
- UDP (messages) and TCP/IP (streams) transport layer protocols used for Internet communication
- External data representation: Serialization/Marshalling and De-serialization/unmarshalling

TCP/IP

- IP: Routing, fragmentation.
  IP-address, CIDR, NAT.
- TCP/UDP: Ports, sockets.
- UDP: Datagrams. Adds checksum to IP.
- TCP: Reliability, congestion control, connections.
  3-way handshake, SYN.
  State machine.

Marshalling

- End-points have to agree on bit-pattern meanings. Typically your programming languages internal representation won’t do.

Models

Models

Models

- Physical models

Models

- Physical models
- Architectural models

Models

- Physical models
- Architectural models
- Fundamental models

Architectural models

Architecture Models

Architecture Models

- Abstraction and patterns for

Architecture Models

- Abstraction and patterns for
  - Entities (objects, components,web-services)

Architecture Models

- Abstraction and patterns for
  - Entities (objects, components,web-services)
  - Communication (interprocess, remote, indirect)

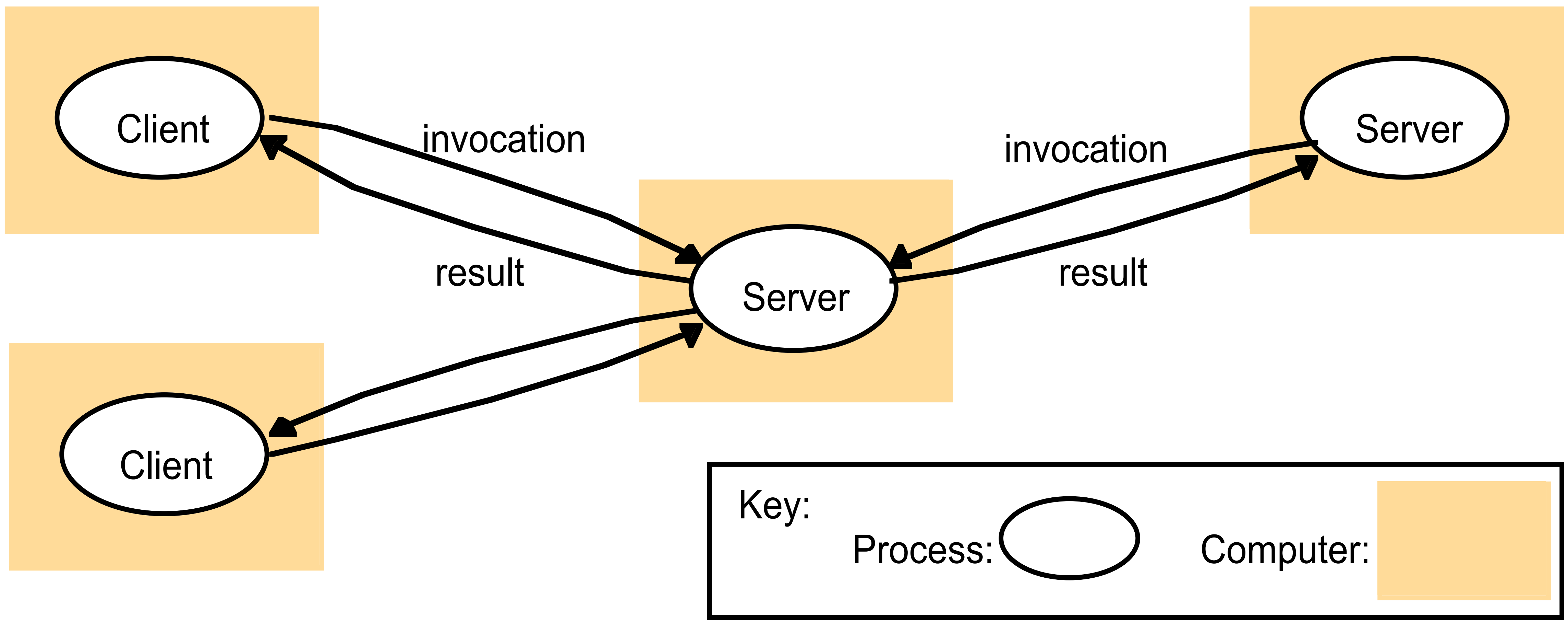
Architecture Models

- Abstraction and patterns for
  - Entities (objects, components,web-services)
  - Communication (interprocess, remote, indirect)
  - Roles and Responsibilities (client, server, peer)

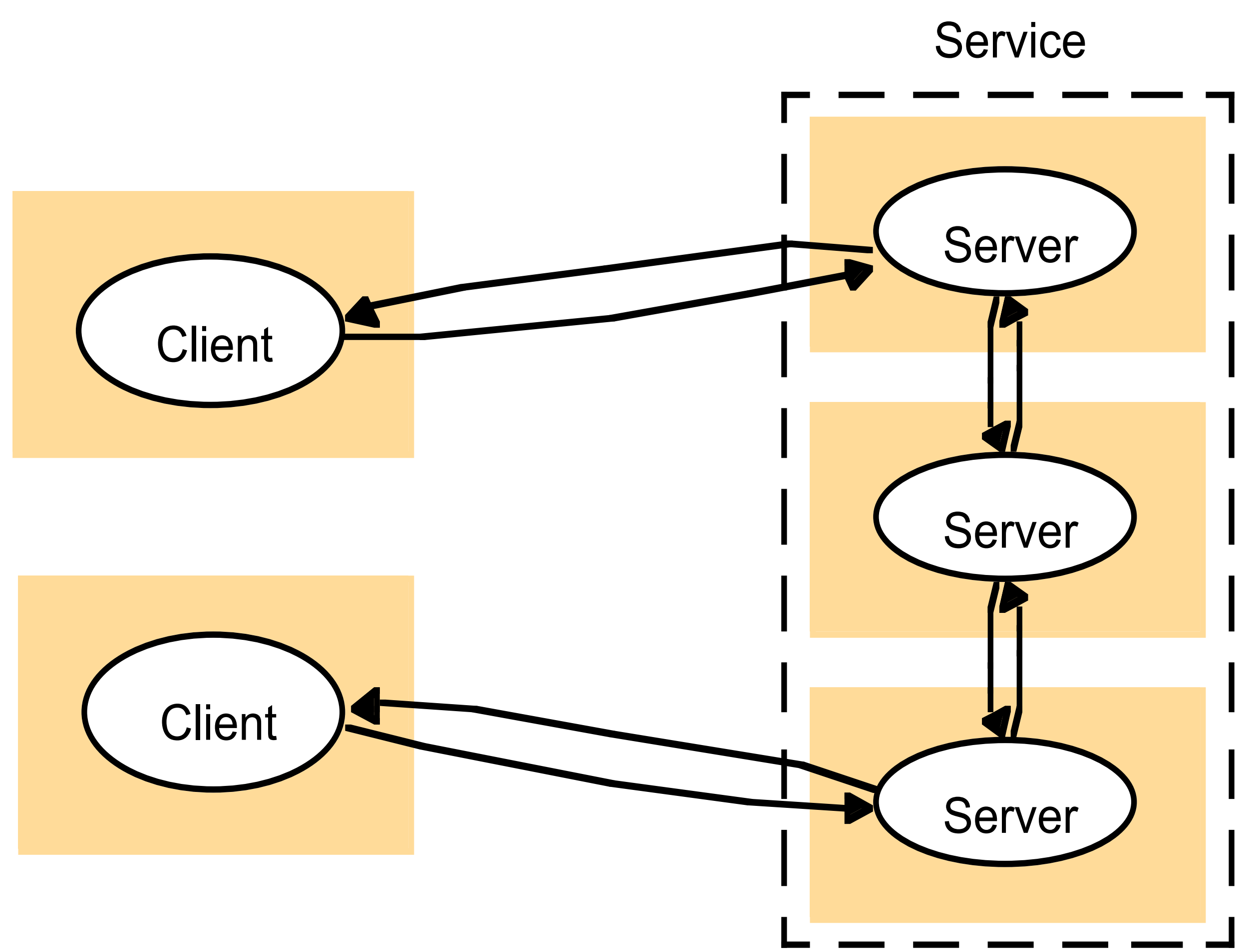
Entities and communication

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Client-server architecture

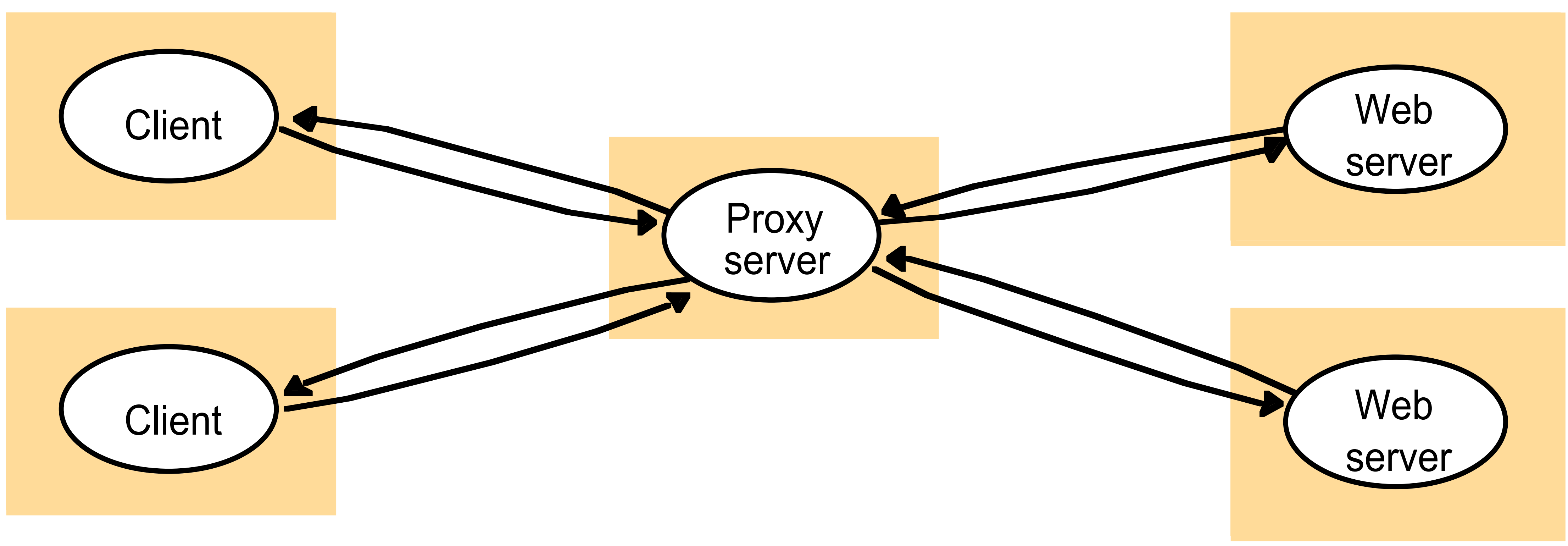


Variation: More servers

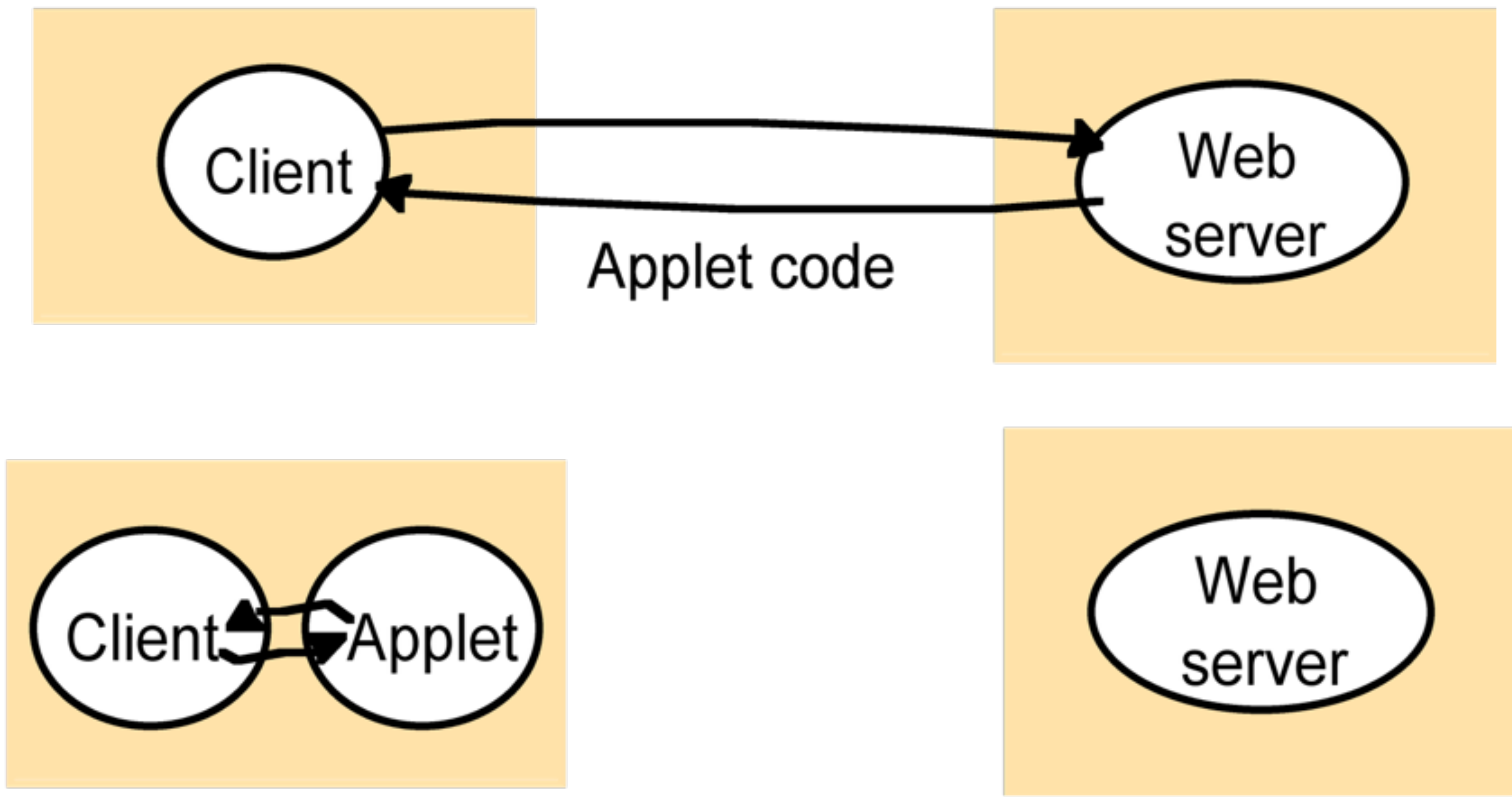


e.g. dividing or sharing (replicated) responsibility

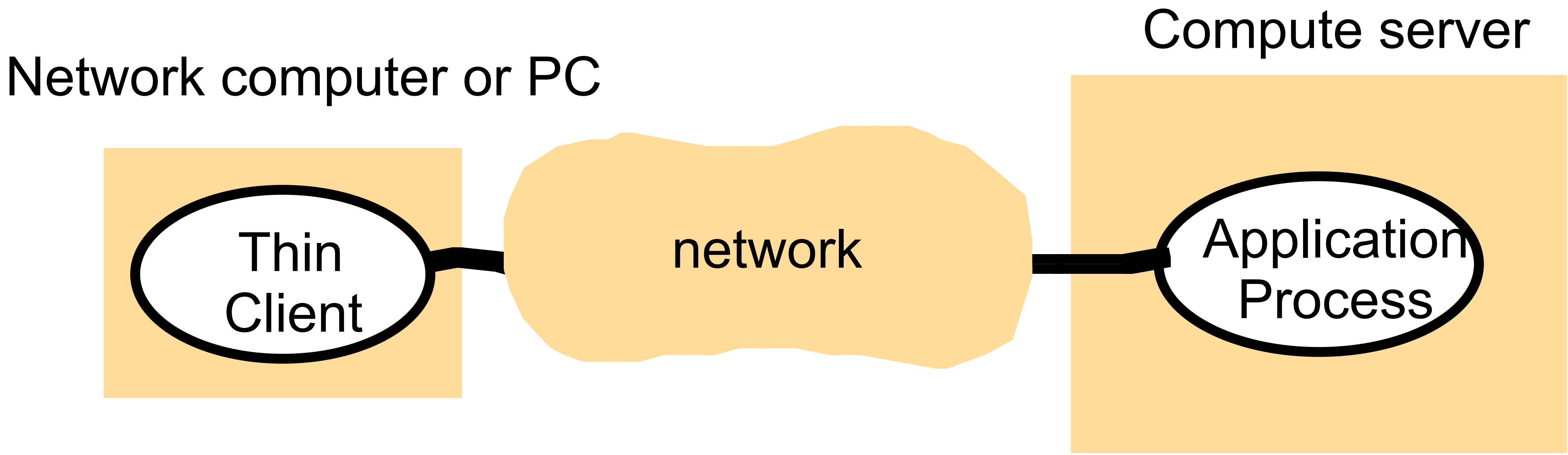
Variation: Proxy & cache



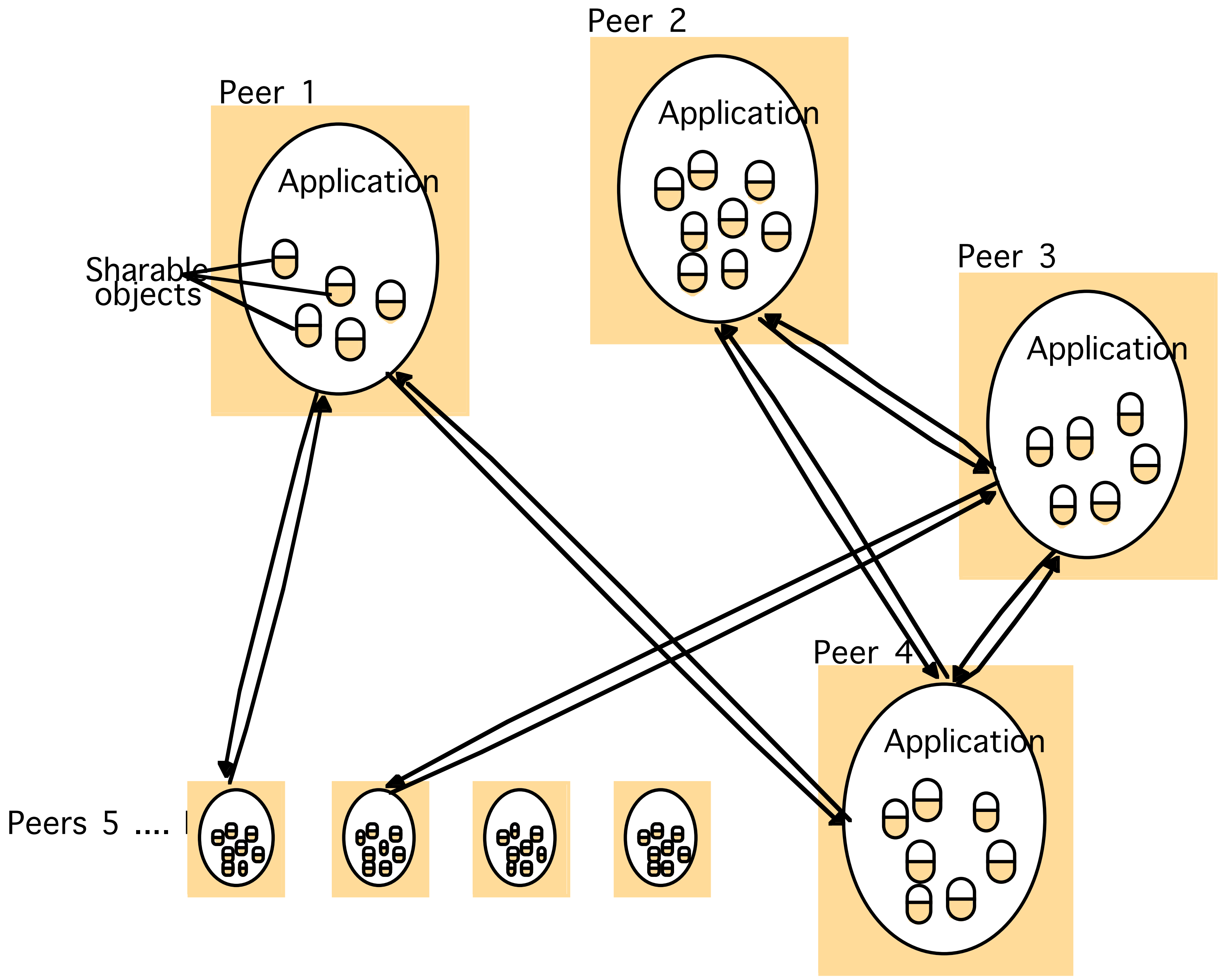
Variation: Mobile code



Variation: Thin clients

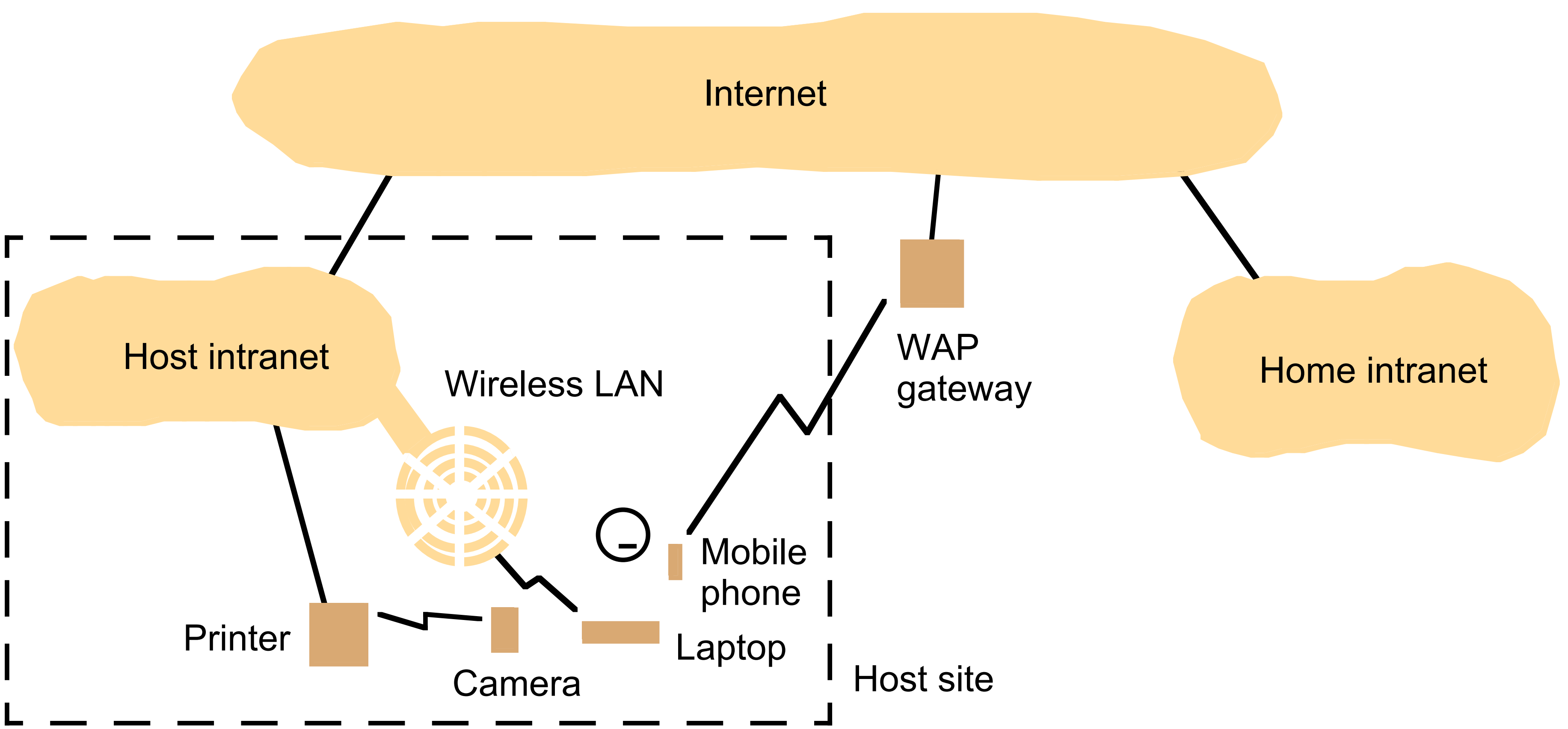


Peer-to-peer



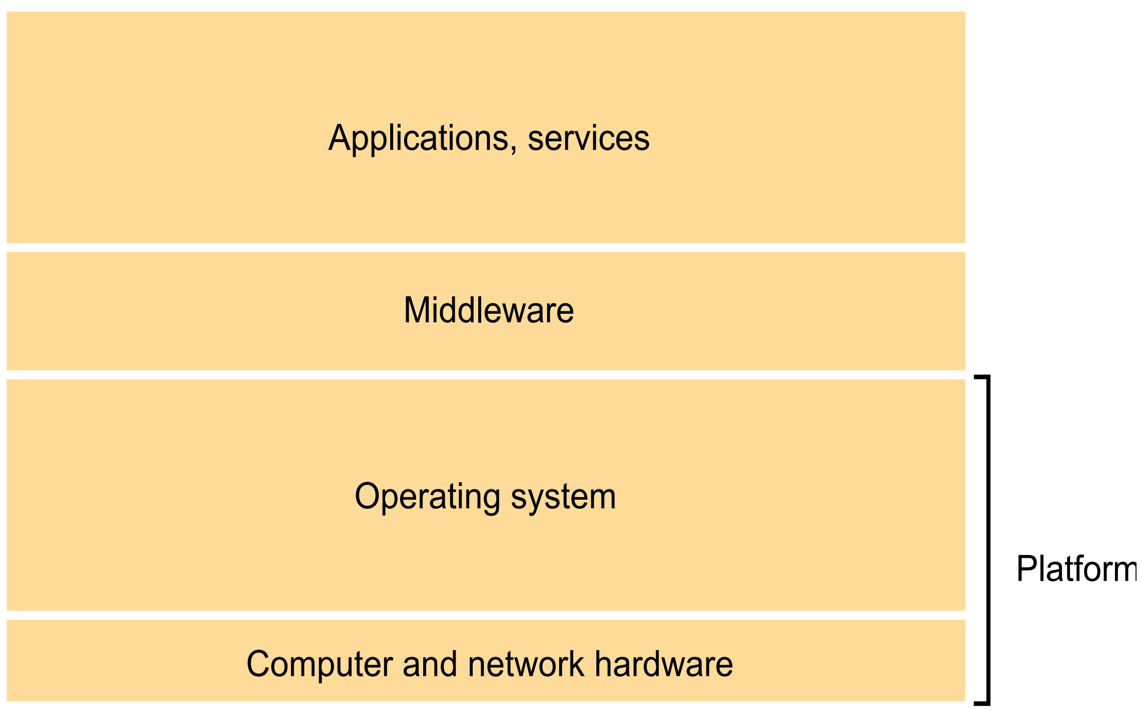
Peer 1
Application
Sharable objects
Peer 2
Application
Peer 3
Application
Peer 4
Application
Peers 5 .... N

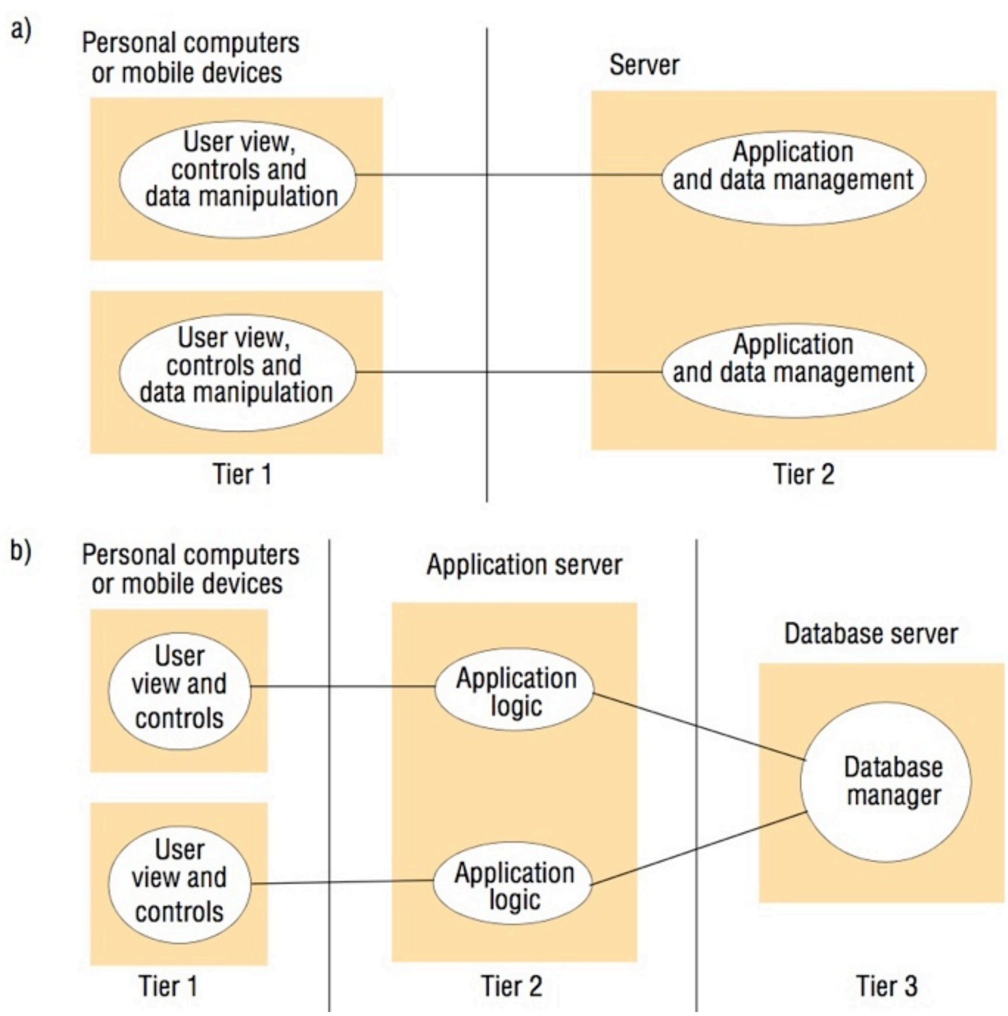
Mobile & adhoc/ spontaneous



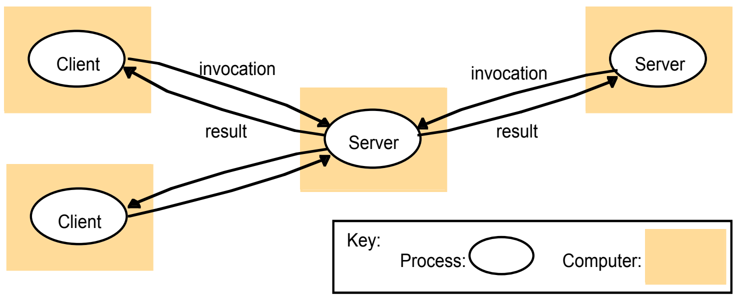
Architectural Patterns

Abstraction vs division of functionality

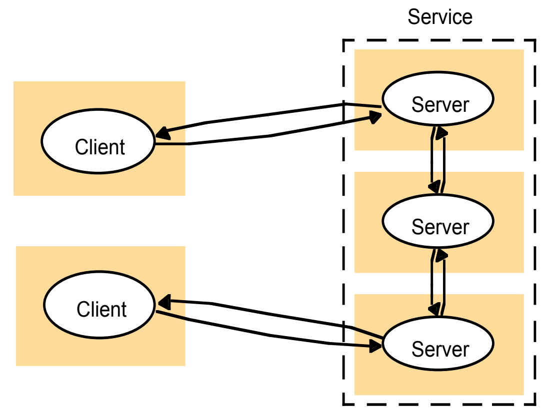




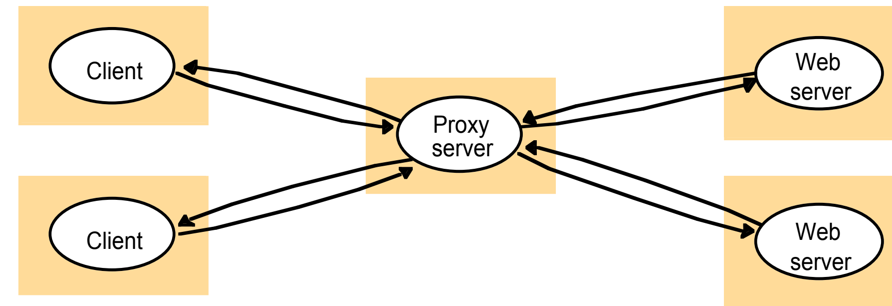
Saltzer's end-to-end argument



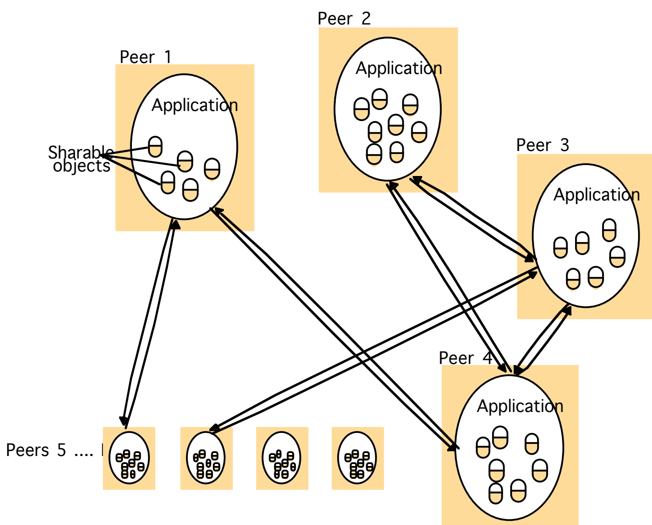
Client/server



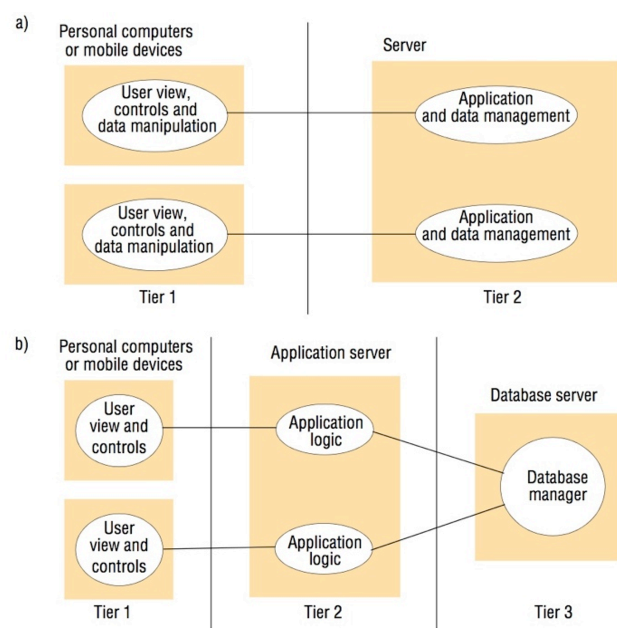
w/Multiple servers



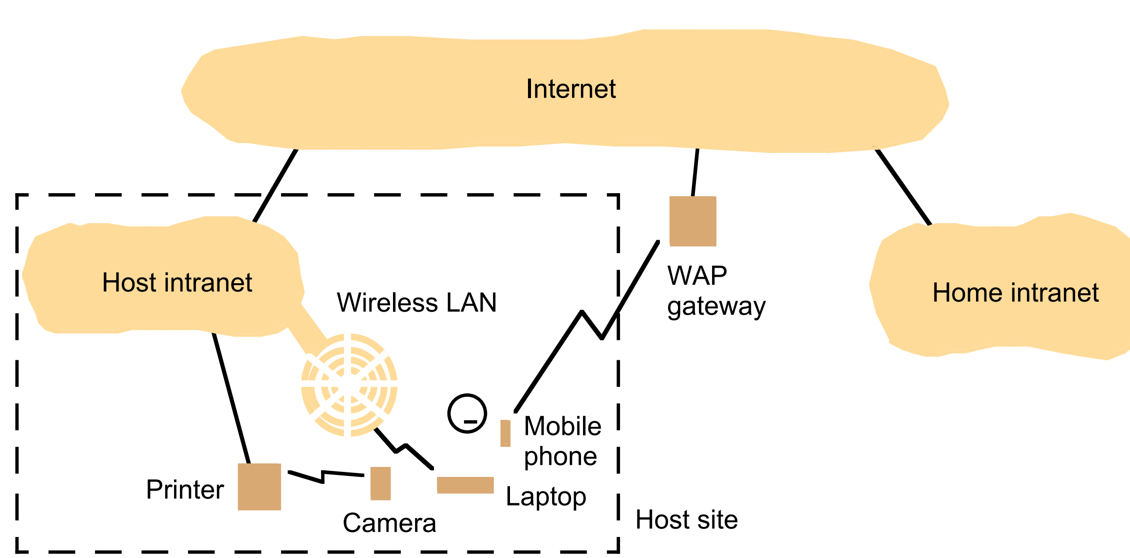
w/Proxies



Peer-to-peer



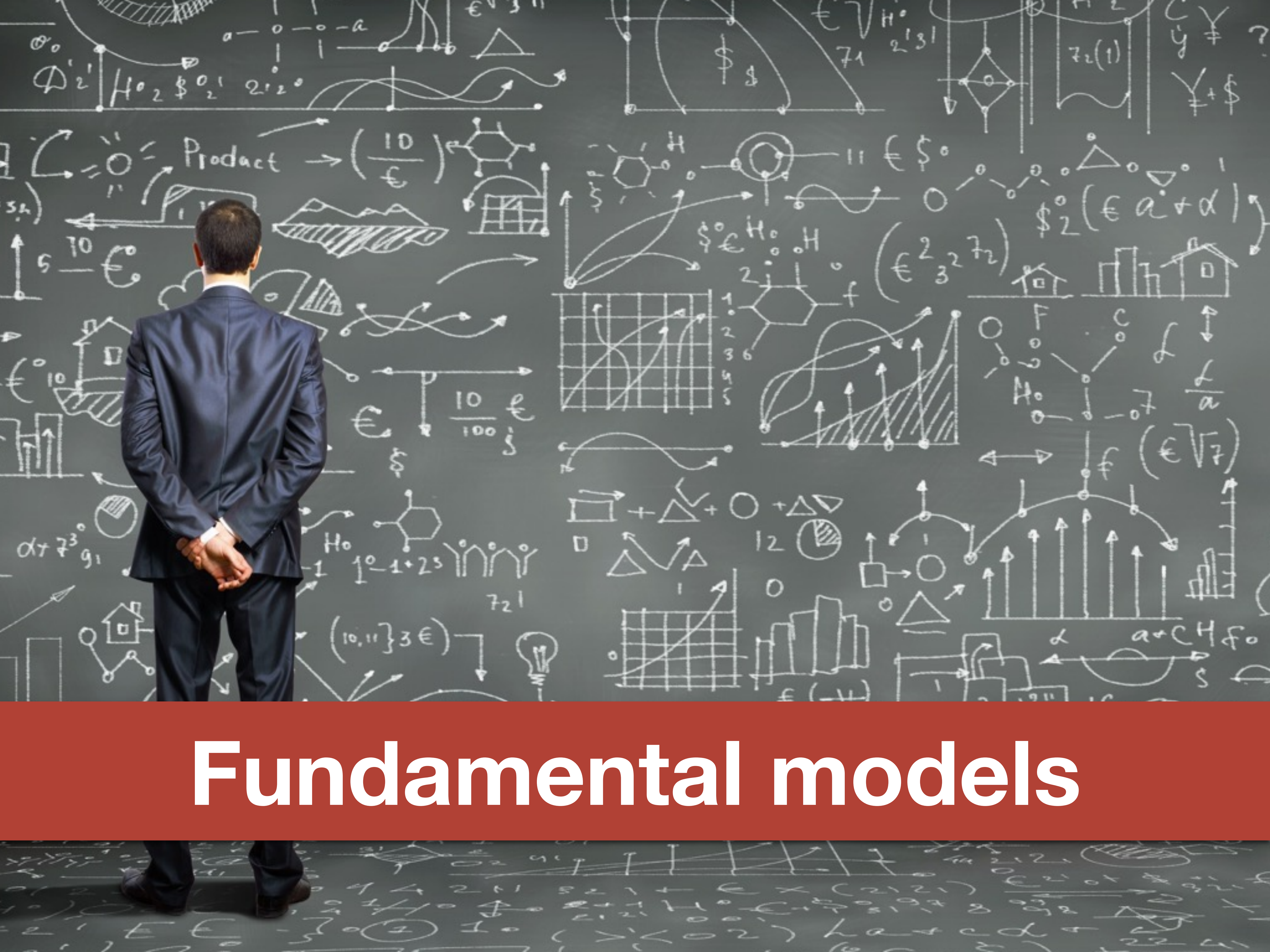
Functionally divided



Mobile/ad hoc

In groups, discuss

What would you imagine is the architectures used by Google Maps, Skype and Netflix?



Fundamental models

Fundamental Models

Abstractions of and assumptions about

- Interaction
- Security
- Failure

Interaction (i)

- latency (from start of transmission to beginning of receipt)
- bandwidth (e.g. bits/second)
- jitter (variation of delivery time)

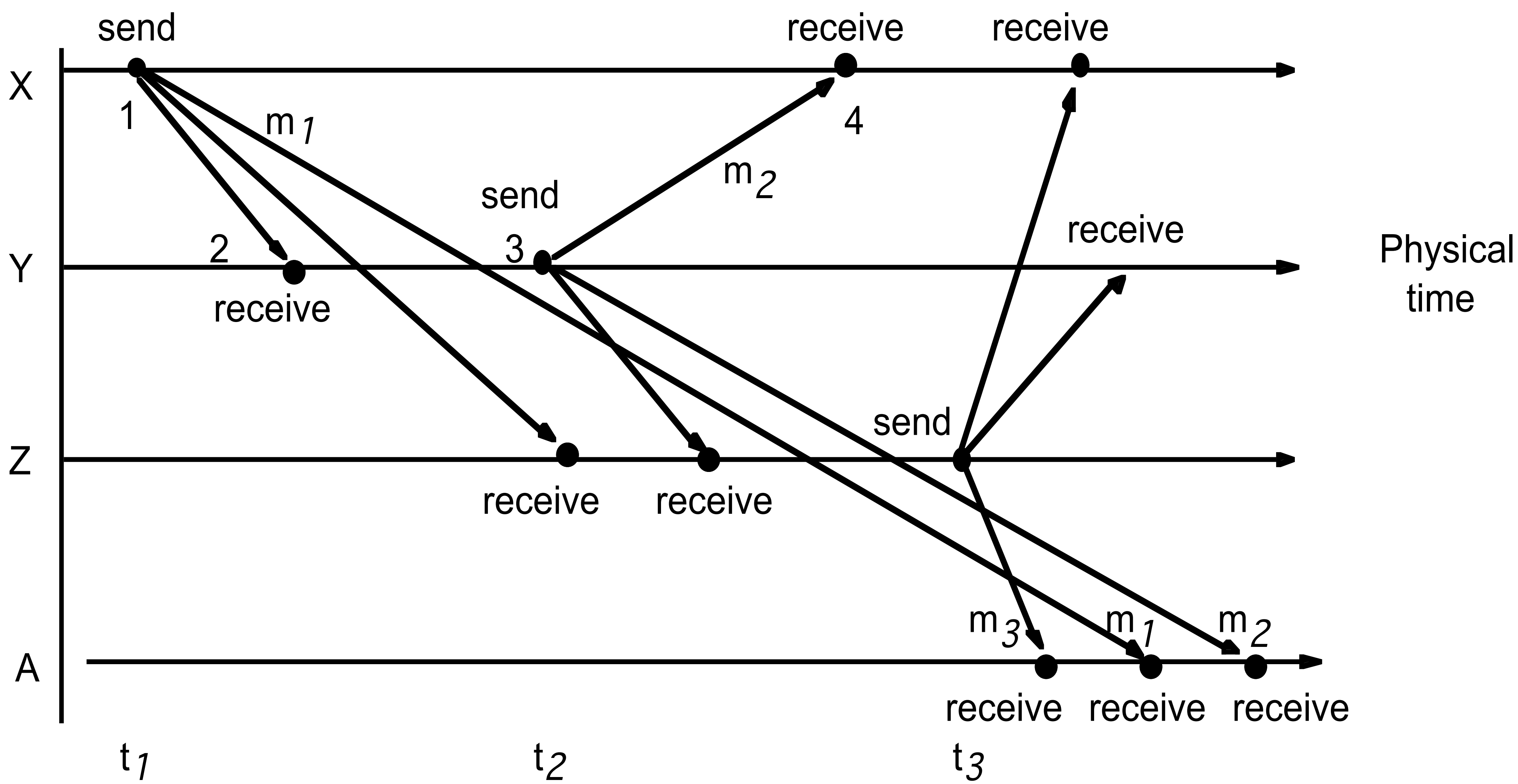
Interaction (ii)

Synchronous

- bounds on execution steps
- guaranteed transmission in bounded time
- clock drift bounds

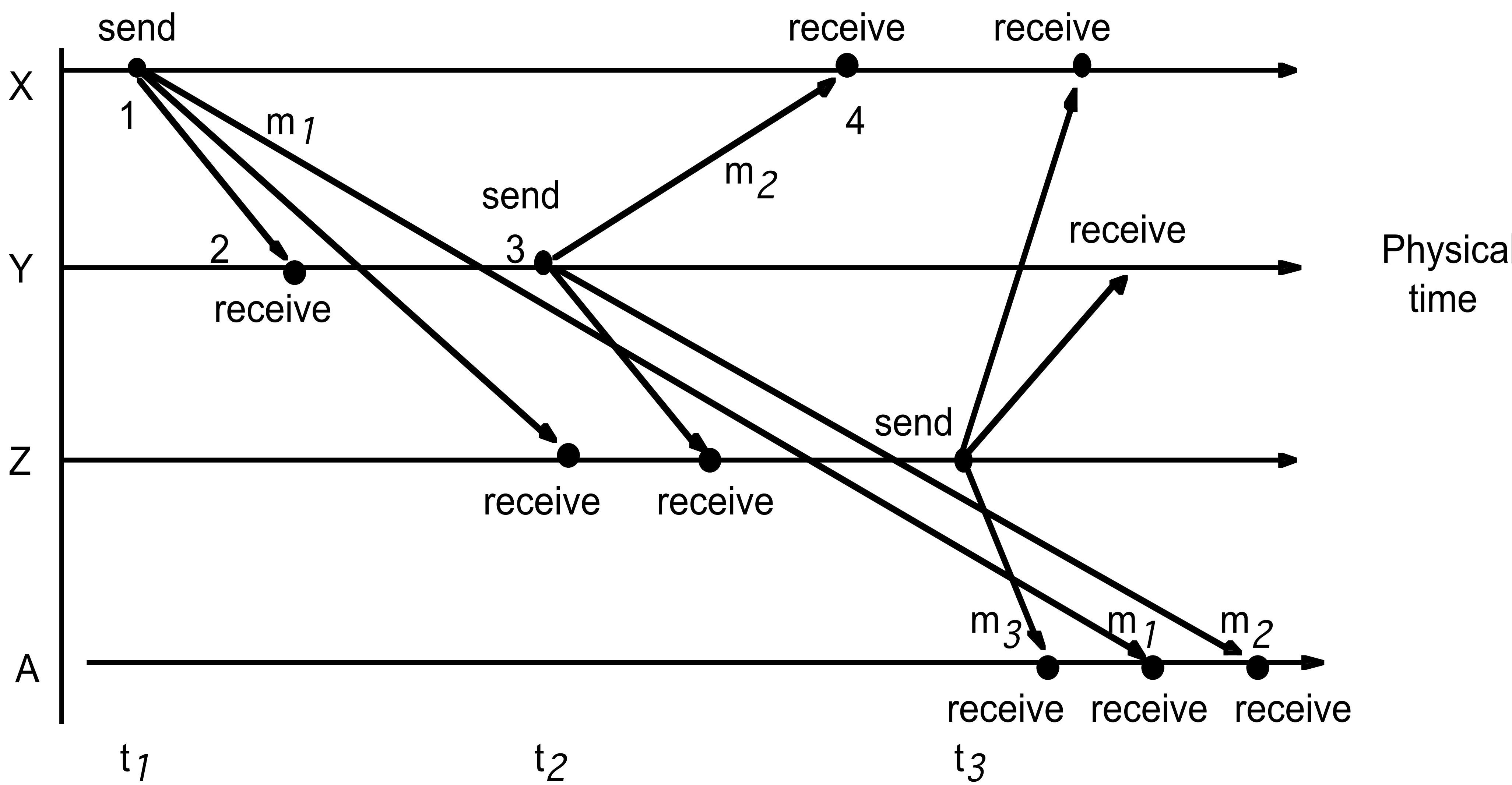
Asynchronous

- no bounds on execution steps
- no time bound on transmission
- arbitrary clock drift



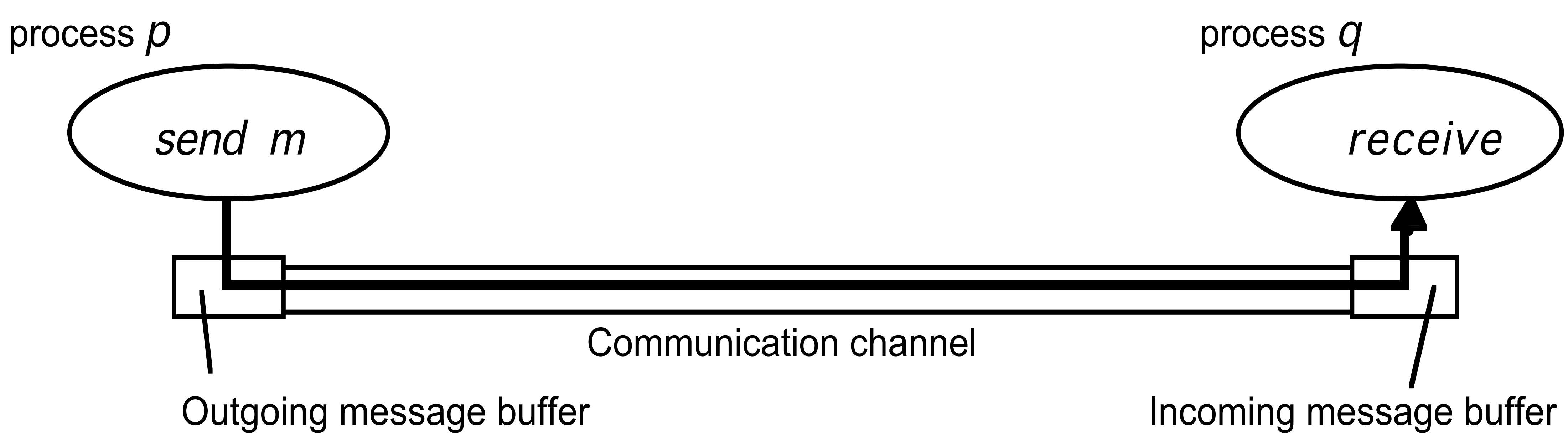
Interaction (iii)

Event ordering—no global time



Discuss in groups

Did A receive m1 before m2?
Did Y or Z receive first?
Did Y send m2 before Z received m1?



Failures (i)

Processes and communication channels

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Failures (ii)

Classification

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Discuss in groups

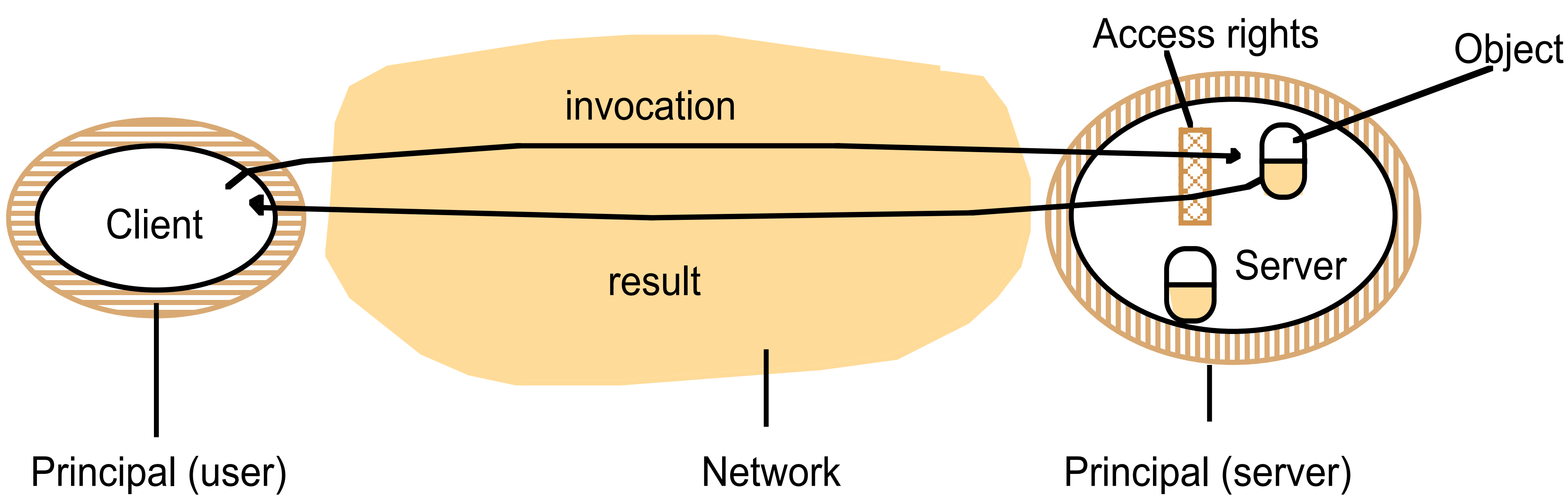
Which failures must Netflix somehow take into account?

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Failures (iii)
(synchronous execution)

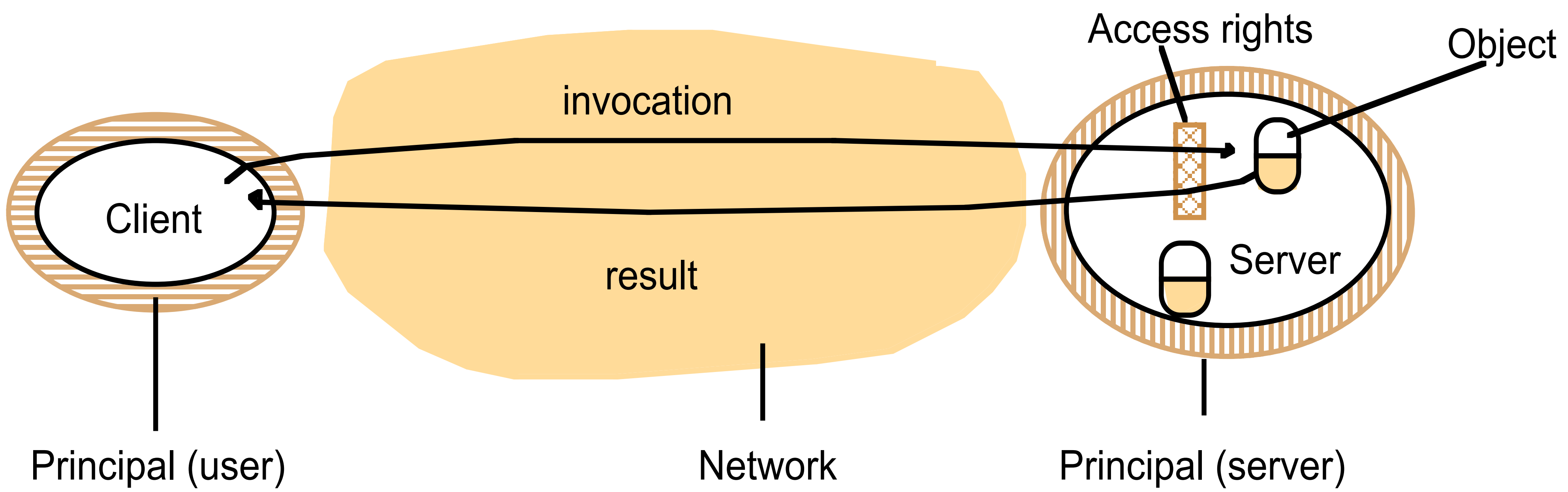
Security Model

- Identification, authentication, confidentiality, integrity
- Threat model: Capabilities of the adversary



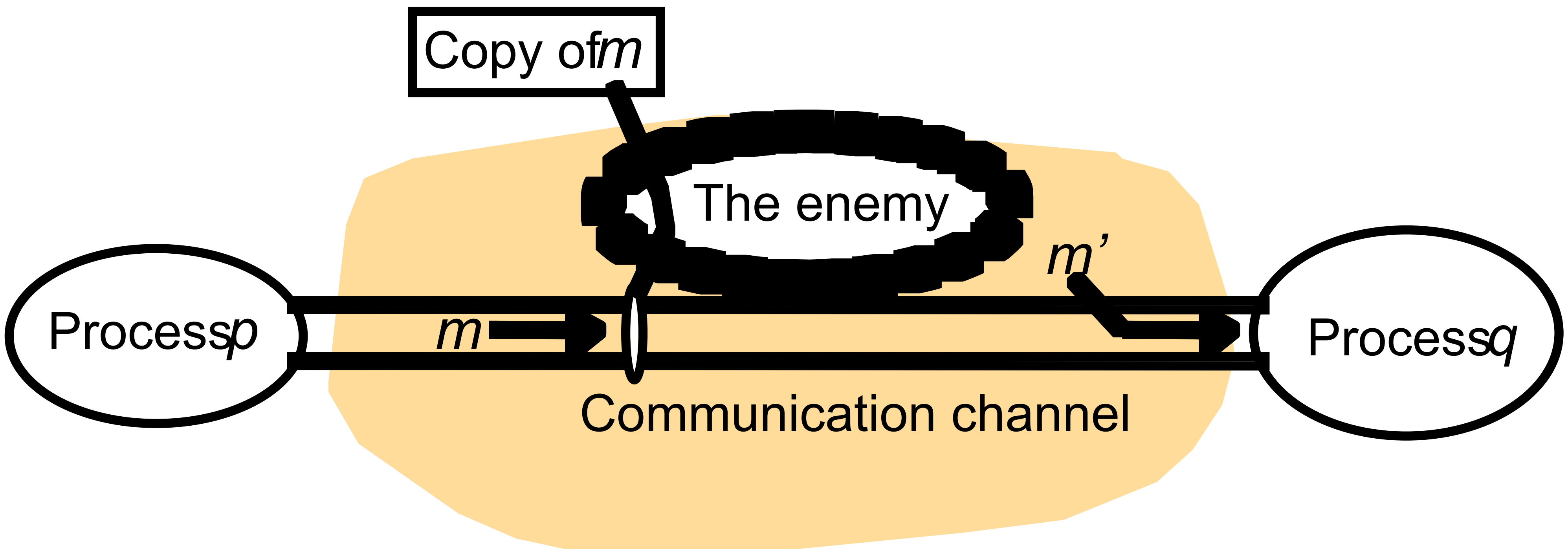
Security Model

- In groups, discuss: What is a reasonable set of capabilities for the adversary?



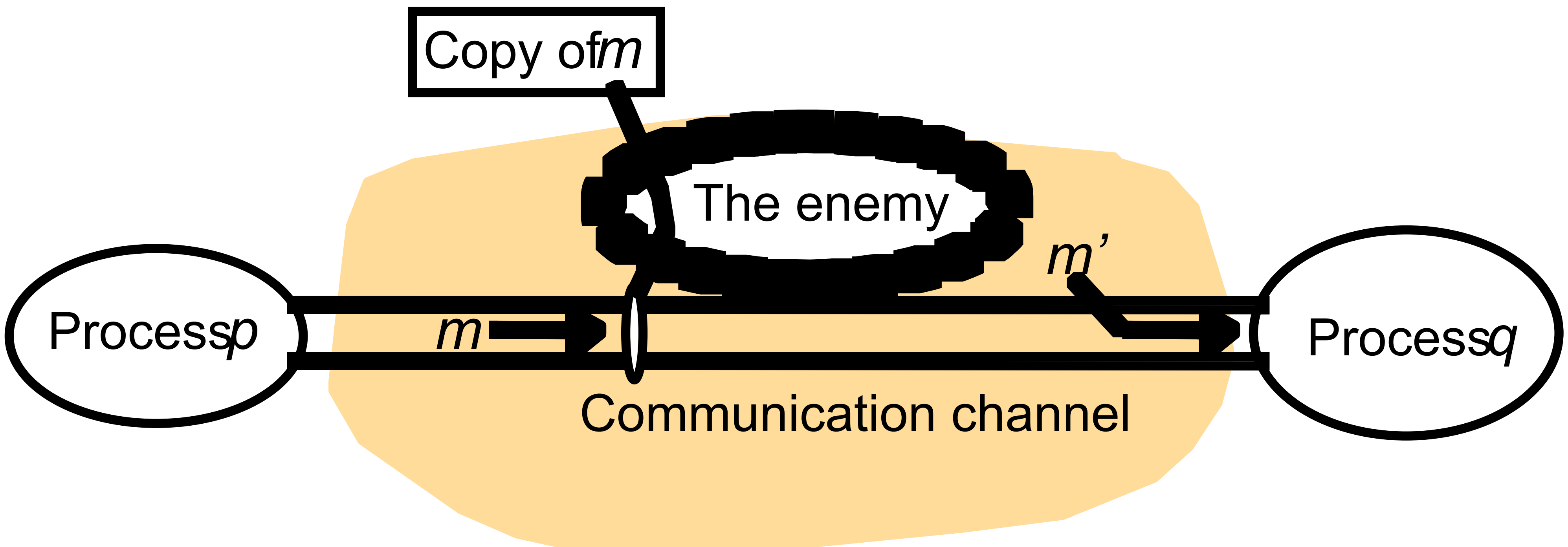
“The adversary”

can copy, remember, modify, add, remove and compose messages



“The adversary”

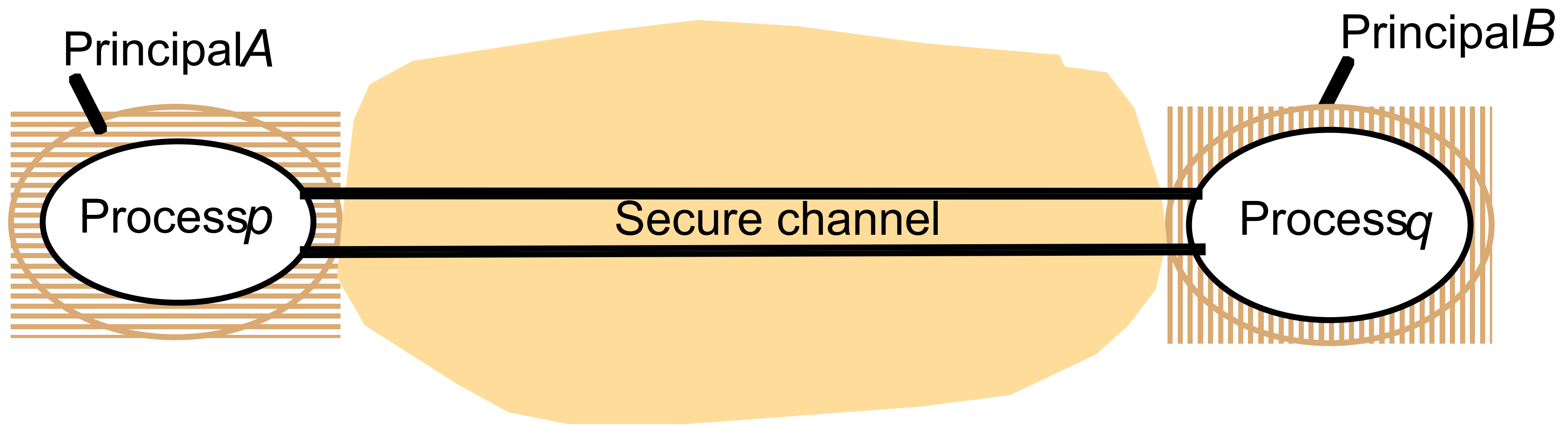
can copy, remember, modify, add, remove and compose messages



but cannot guess your secrets (e.g. keys)
- unless given enough time!

Security handling

- Cryptography, authentication, secure channels (SSL)



Other threats

- Denial of service
- Mobile code

Summary

- **Physical models:** local heterogenous to global/ ubiquitous/pervasive computing

- **Architecture models:**

  Patters for distribution and interaction of components: Client-server, peer-to-peer, layers, tiers

- **Fundamental models:**

  Interaction, failure and security

Mini-project 1

Good.

- Most groups gave reasonable solutions to all 4 questions.
- Some problems with threads.
- Many groups forgot either corruption or duplication in Question 4.
- All groups wrote servers that use O(n) space in Question 4.

The process

- TAs says ~50 people come to TA sessions. More, please.
- If TAs says “come talk to us” in feedback, please do.
- Do use the forum.
- Do read the hints.
- **Important!** 90 people submitted. If you didn’t, and still want to follow the course, come see me.
- Please fill out survey on learnit.

What do you think?


Well done

What was the point?

What was the point?

- Understand UDP, IP, Networking by working with it.

What was the point?

- Understand UDP, IP, Networking by working with it.
- Get a sense of how difficult getting a message across reliably is.

What was the point?

- Understand UDP, IP, Networking by working with it.
- Get a sense of how difficult getting a message across reliably is.
- Understand the basic challenge of distributed systems: Channel & Crash failures.

What was the point?

- Understand UDP, IP, Networking by working with it.
- Get a sense of how difficult getting a message across reliably is.
- Understand the basic challenge of distributed systems: Channel & Crash failures.
- Get a practical foundation for understanding the Model's chapter of the book.

Reliable transfers

Reliable transfers

- Print-message might be lost.

Reliable transfers

- Print-message might be lost.
- Must detect channel's omission failure.

Reliable transfers

- Print-message might be lost.
- Must detect channel’s omission failure.
- Server sends acknowledgment,
  client times out and resends print-message on missing acknowledgment.

Acknowledgment lost

Acknowledgment lost

- What if the acknowledgment is lost?

Acknowledgment lost

- What if the acknowledgment is lost?
  - Client retransmits Print message.

Acknowledgment lost

- What if the acknowledgment is lost?
  - Client retransmits Print message.
  - Server prints /twice/.

Acknowledgment lost

- What if the acknowledgment is lost?
  - Client retransmits Print message.
  - Server prints /twice/.
- What if the Print message is duplicated?

Acknowledgment lost

- What if the acknowledgment is lost?
  - Client retransmits Print message.
  - Server prints /twice/.
- What if the Print message is duplicated?
  - Server prints /twice/.

Acknowledgment lost

- What if the acknowledgment is lost?
    - Client retransmits Print message.
    - Server prints /twice/.
- What if the Print message is duplicated?
    - Server prints /twice/.
- Key problem: Server can’t tell if Print message is duplicated or two genuine requests to print the same thing.

Duplicates

Duplicates

- Client invents GUID.
  C: PRINT(GUID, ‘Hello world’)
  S: OK(GUID)

Duplicates

- Client invents GUID.
  C: PRINT(GUID, ‘Hello world’)
  S: OK(GUID)

- Server can check on receiving PRINT(GUID, …) whether it already printed that GUID.

Duplicates

- Client invents GUID.
  C: PRINT(GUID, ‘Hello world’)
  S: OK(GUID)

- Server can check on receiving PRINT(GUID, …) whether it already printed that GUID.

- NB! Timestamps aren’t GUIDs.

Duplicates

- Client invents GUID.
  C: PRINT(GUID, ‘Hello world’)
  S: OK(GUID)

- Server can check on receiving PRINT(GUID, …) whether it already printed that GUID.

- NB! Timestamps aren’t GUIDs.

  - Somebody else might be printing the same message at the same (local) time.

Duplicates

- Client invents GUID.
  C: PRINT(GUID, ‘Hello world’)
  S: OK(GUID)

- Server can check on receiving PRINT(GUID, …) whether it already printed that GUID.

- NB! Timestamps aren’t GUIDs.

  - Somebody else might be printing the same message at the same (local) time.

  - Also, no global time.

What about resources?

What about resources?

- How much space does the server use?

What about resources?

- How much space does the server use?
- O(n) where n is the number of messages printed. I.e., the server will eventually run out of space.

What about resources?

- How much space does the server use?
- O(n) where n is the number of messages printed. I.e., the server will eventually run out of space.
- Can we do better?

Sessions

Sessions

- Server: Don’t register what you did, register what you must do.

Sessions

- Server: Don’t register what you did, register what you must do.
- So, sessions. Server assigns session-ids.

Sessions

- Server: Don’t register what you did, register what you must do.
- So, sessions. Server assigns session-ids.
- Sequence is now:
  C: READY?
  S: READY(1)   <— Reserve spot 1
  C: PRINT(1, “Hello world”)
  S: OK(1)   <— Delete spot 1

Sessions

- Server: Don’t register what you did, register what you must do.
- So, sessions. Server assigns session-ids.
- Sequence is now:
  C: READY?
  S: READY(1) <— Reserve spot 1
  C: PRINT(1, “Hello world”)
  S: OK(1) <— Delete spot 1
- Server can now distinguish duplicates:

Sessions

- Server: Don’t register what you did, register what you must do.
- So, sessions. Server assigns session-ids.
- Sequence is now:
  C: READY?
  S: READY(1)   <— Reserve spot 1
  C: PRINT(1, “Hello world”)
  S: OK(1)   <— Delete spot 1
- Server can now distinguish duplicates:
  - On receive PRINT(n, msg):
        If “Reserve spot n”, print msg, “delete spot n”.
        else we already printed n.
    In either case, send OK(n).

Remaining problems

- What if READY(n) is lost?
- What if the Client suffers a crash failure?

READY(n) lost

- Add a step.
  C: READY?
  S: READY(n)
  C: CONFIRM(n)   <— Server reserves slot(n)

Client crashes

- Solution A. Assume client completes protocol when he comes back up.

- “Solution” B. Timeouts.
  In case of long delays, client will think his message was lost.

Thank you!

Remember the poll at:

https://learnit.itu.dk/mod/choice/view.php?id=50344