Internet Services

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Internet and Web Architecture

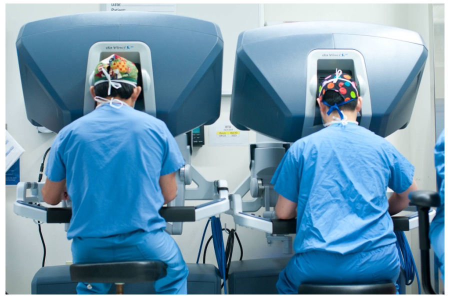
Lecture 1

Srinivas Narayana

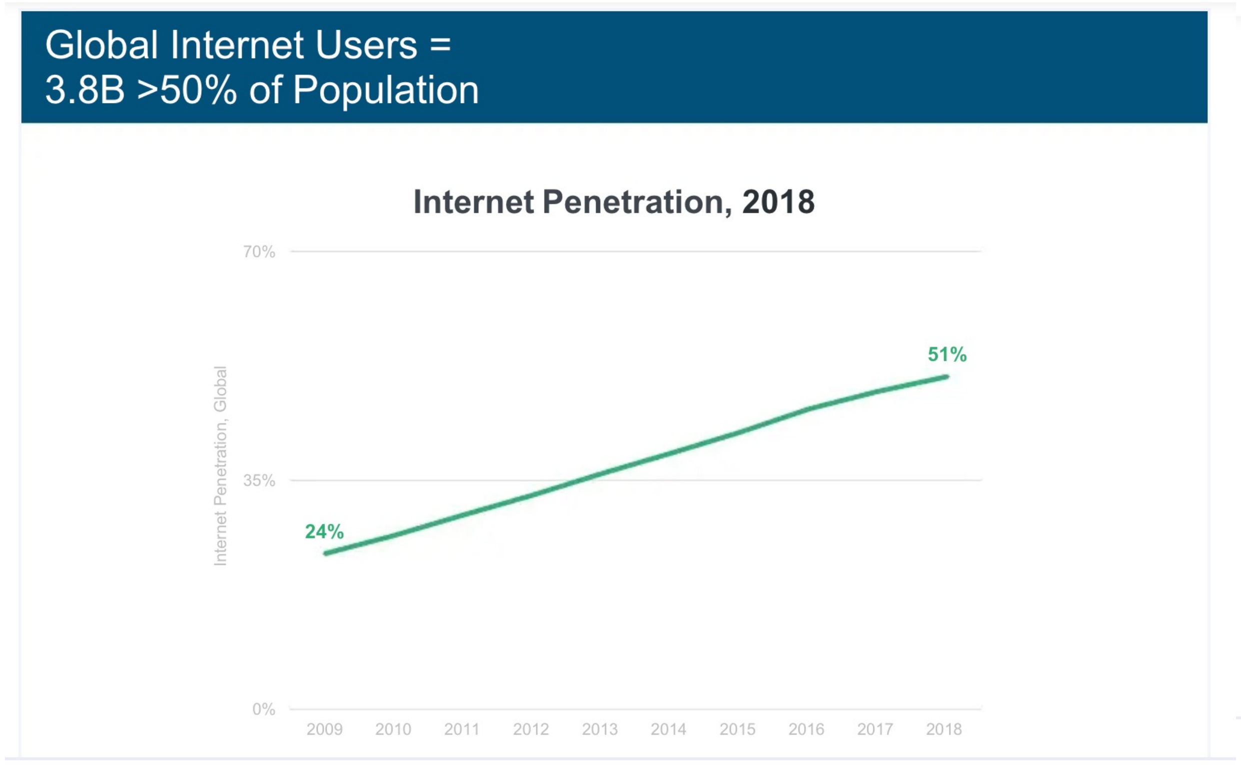
http://www.cs.rutgers.edu/~sn624/553-S23

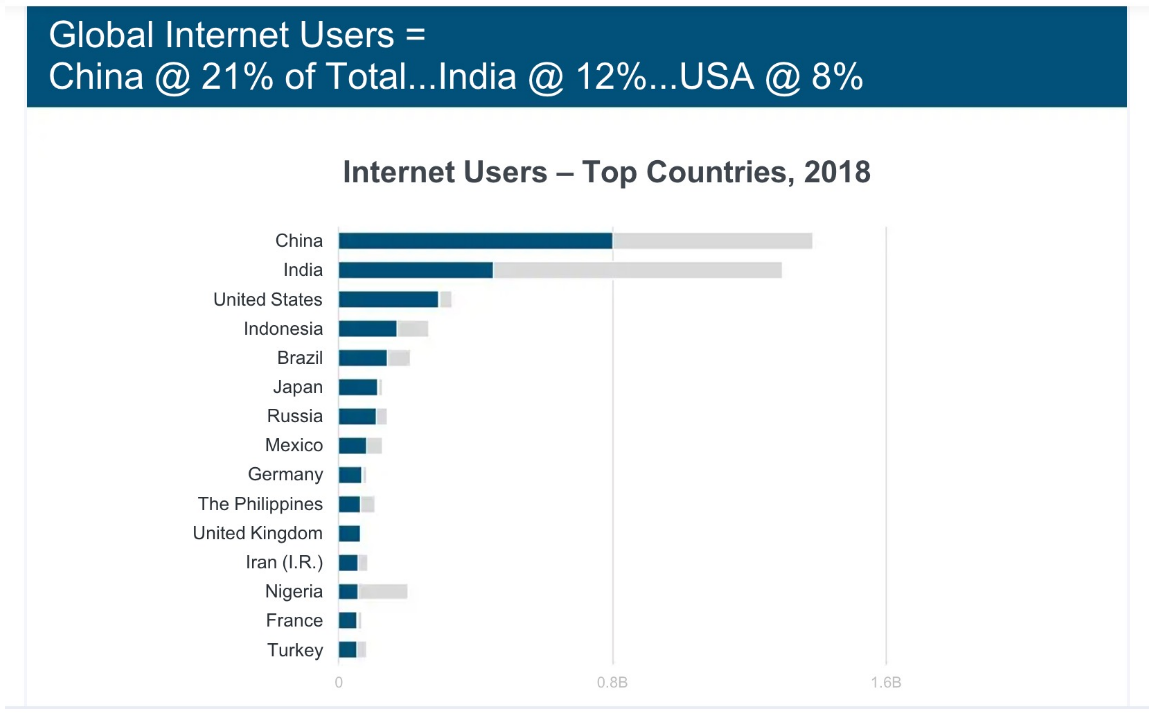
RUTGERS
UNIVERSITY | NEW BRUNSWICK

Our life on the Internet



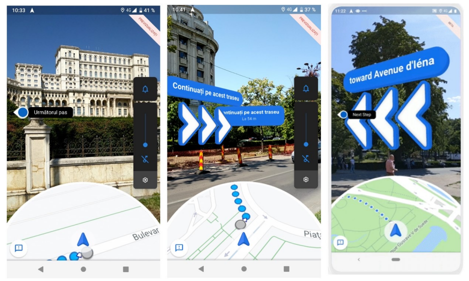
Internet users are everywhere

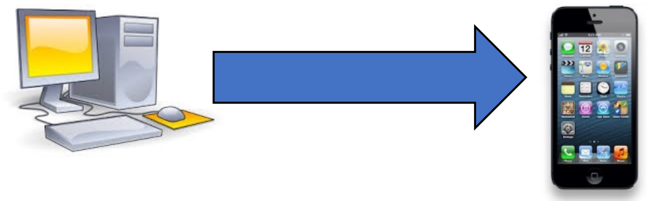




Source: Mary Meekers, 2019 Internet trends

Evolution of Internet services





1992	1996	2000	2004	2008
ftp	chat	news	Music	Wikipedia
Web	Games	Blog	itunes	Craiglist
email	IM	Search	Games	Youtube
	Yahoo!		search	

2010-2020

2020--

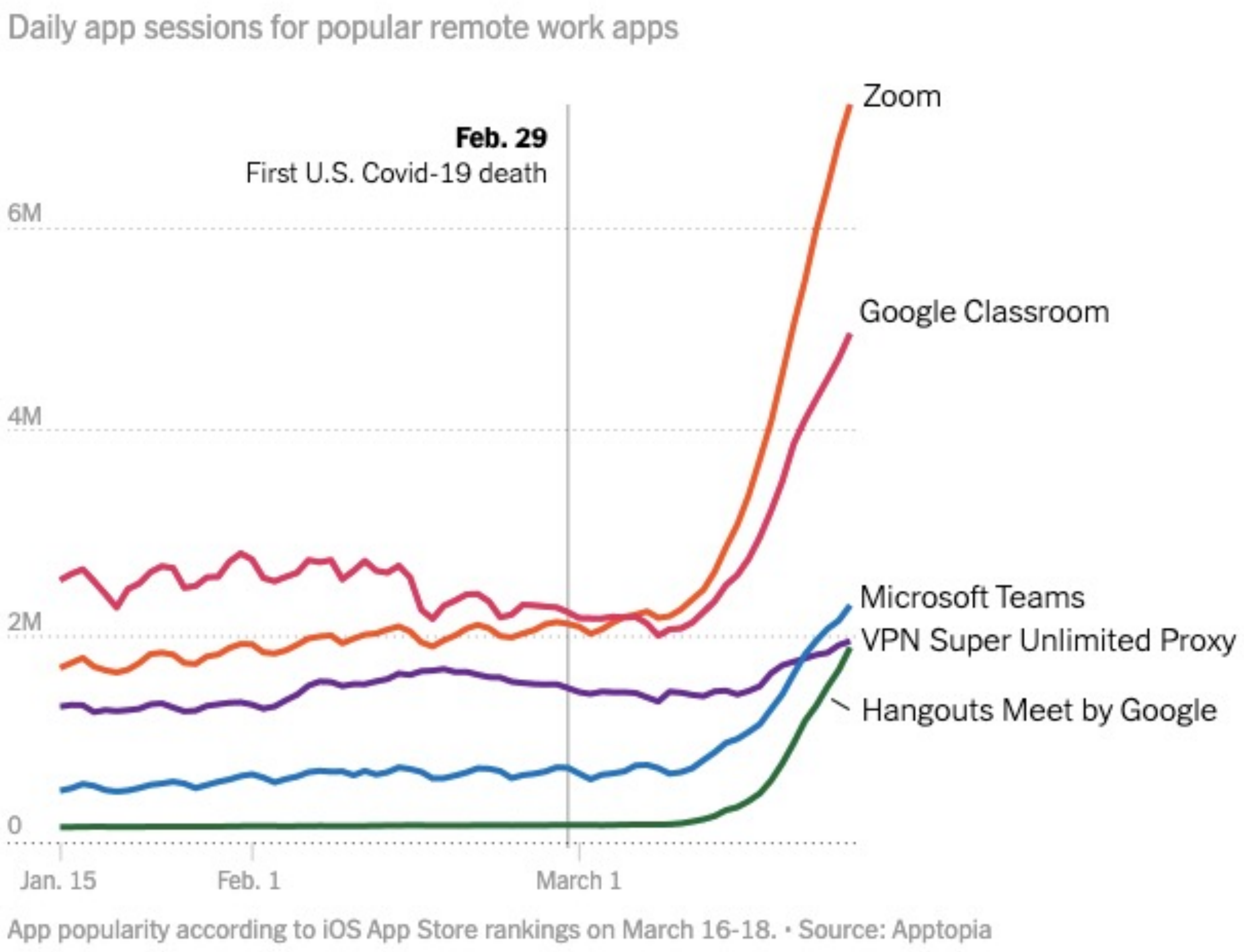
Text-heavy

Multimodal media
User-generated
content

Augment physical world

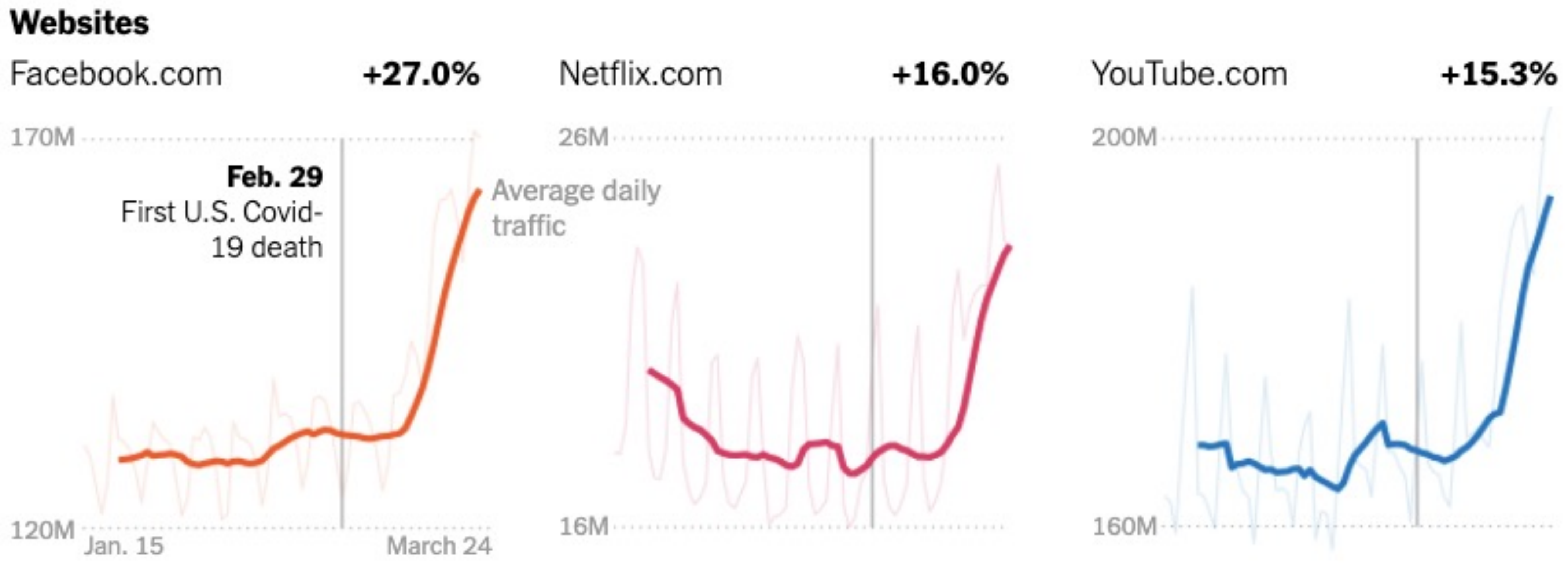
Replace phy world

Pandemic shifts: how we worked



Data shows number of daily sessions in the US over a period in 2020. Source: nytimes

… and played



Data shows number of daily sessions in the US over a period in 2020. Source: nytimes

Internet Services

Understand how modern Internet services designed

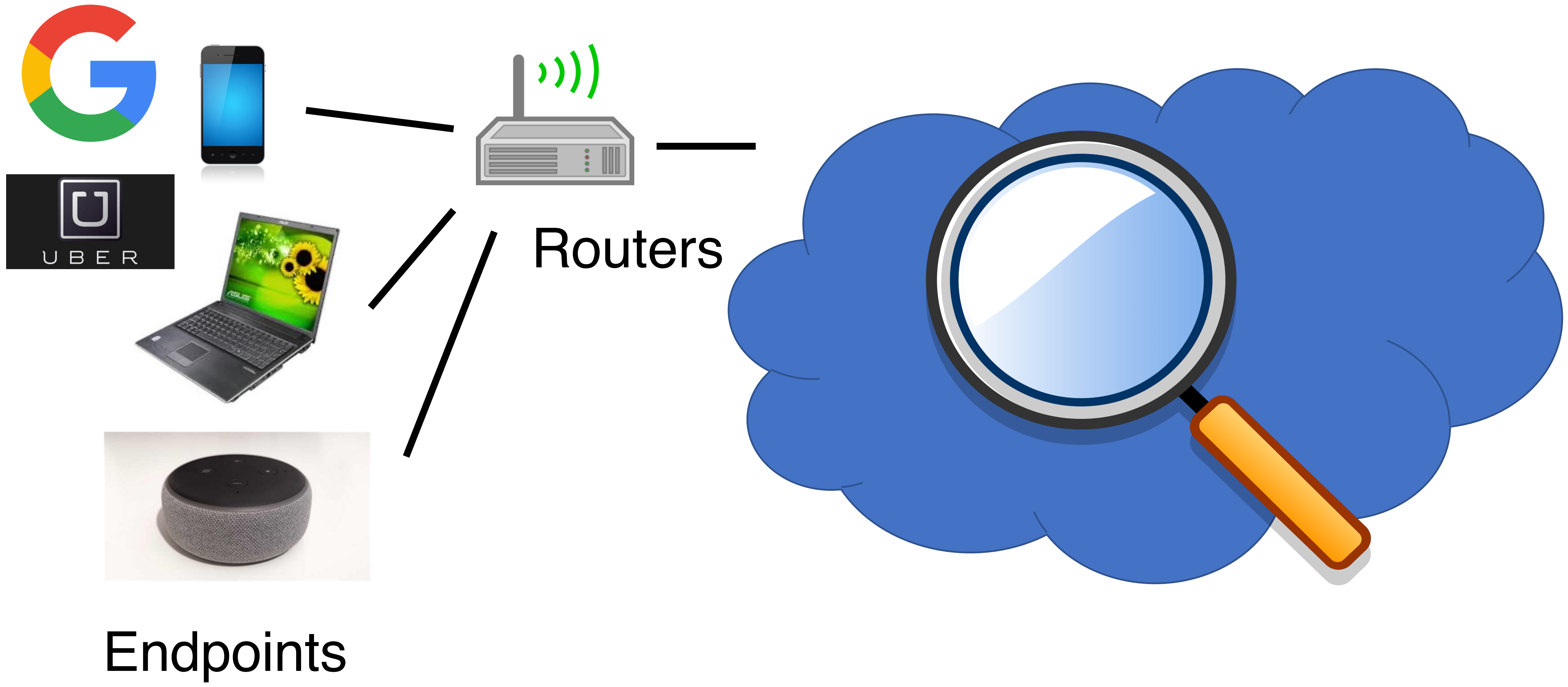
Learn fundamental concepts in application, system, and network design that make them possible

Practice your knowledge

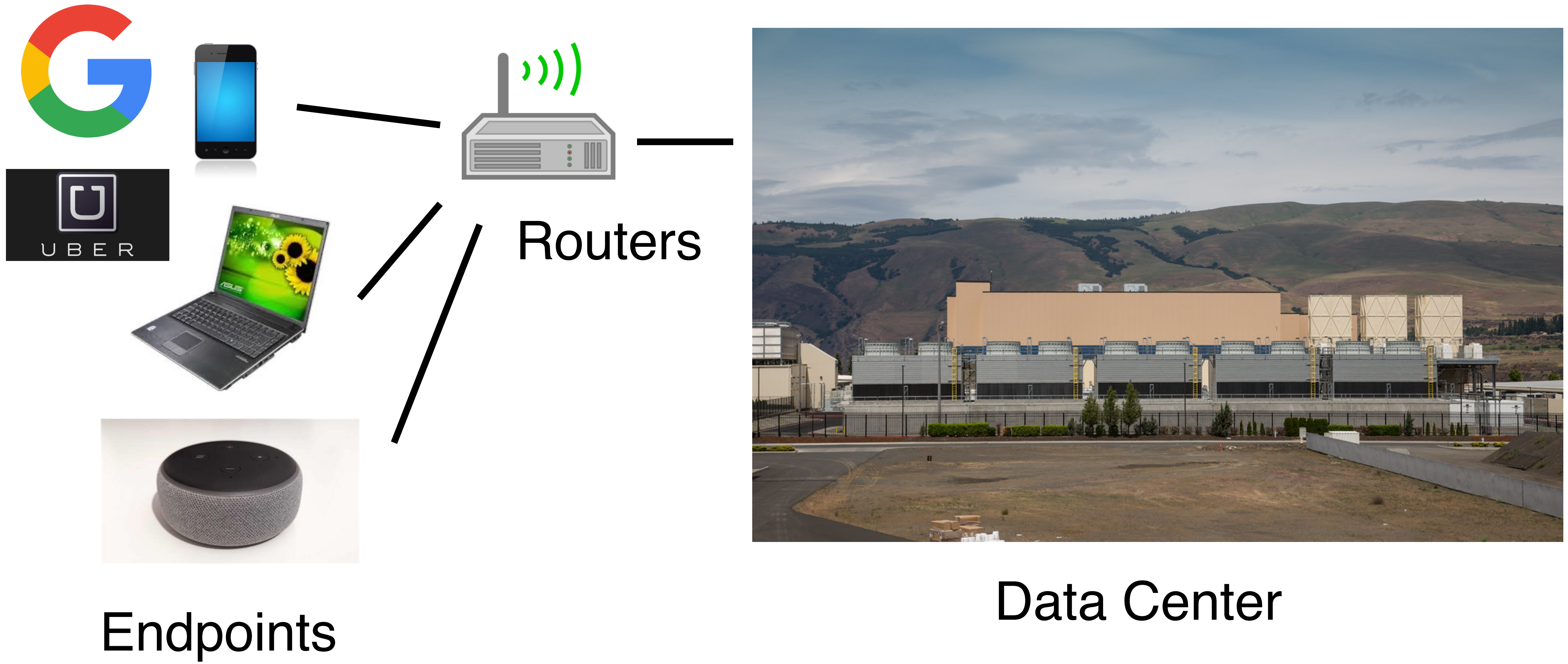
What is an Internet service made of?

Applications, Endpoints, and interconnecting Network

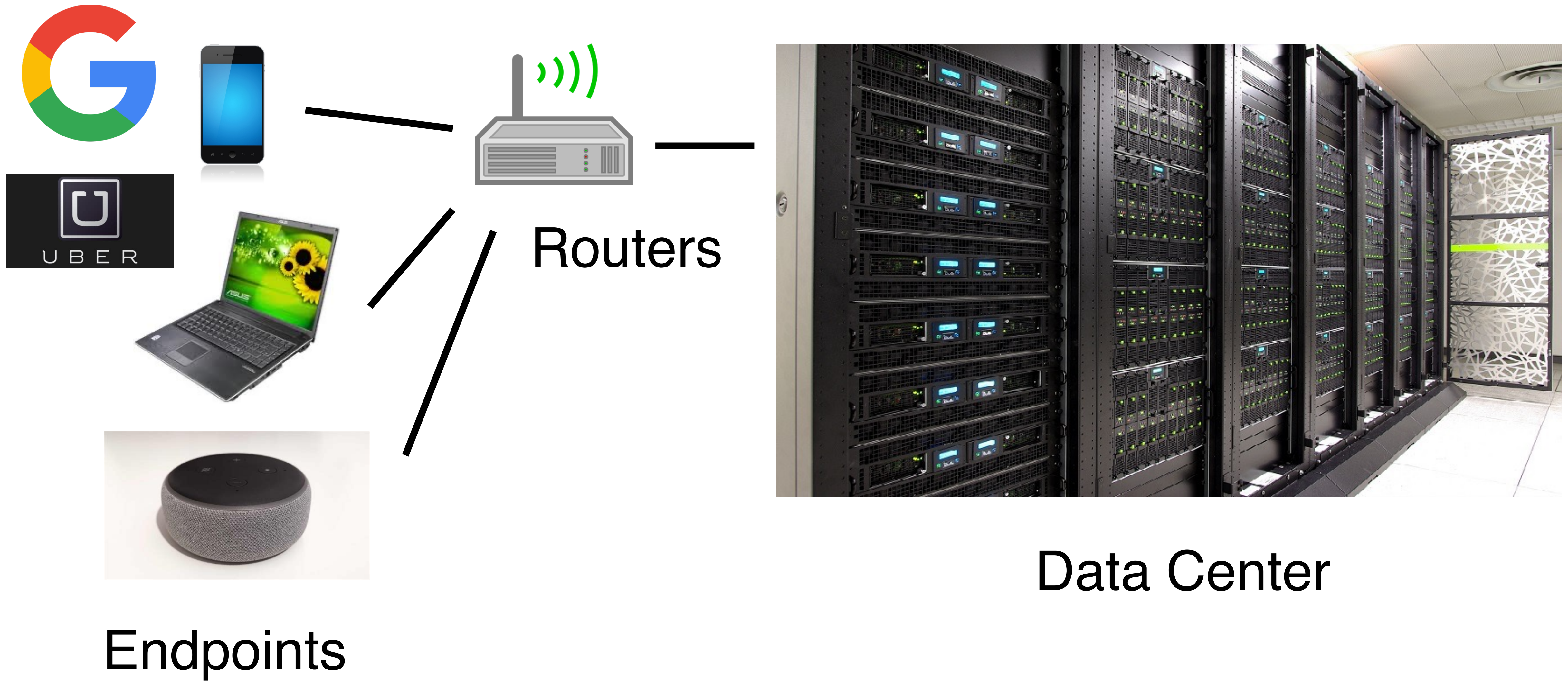
Components of an Internet Service



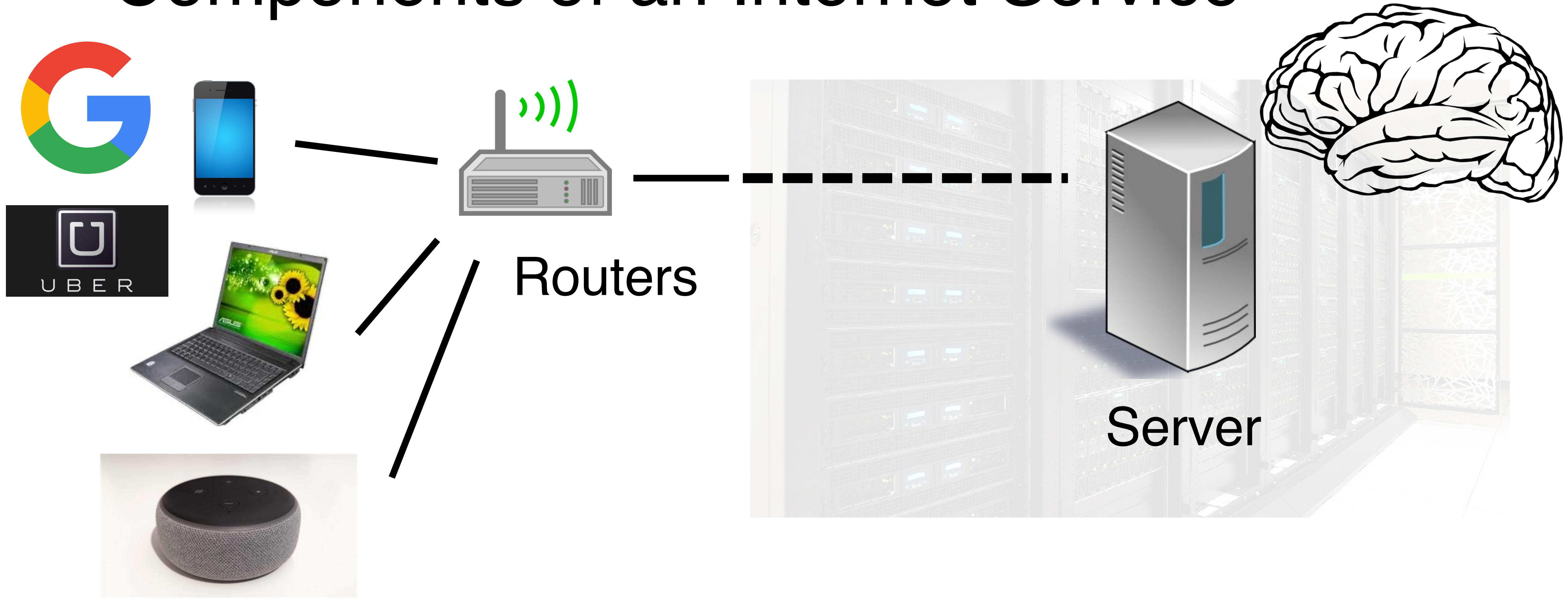
Components of an Internet Service



Components of an Internet Service



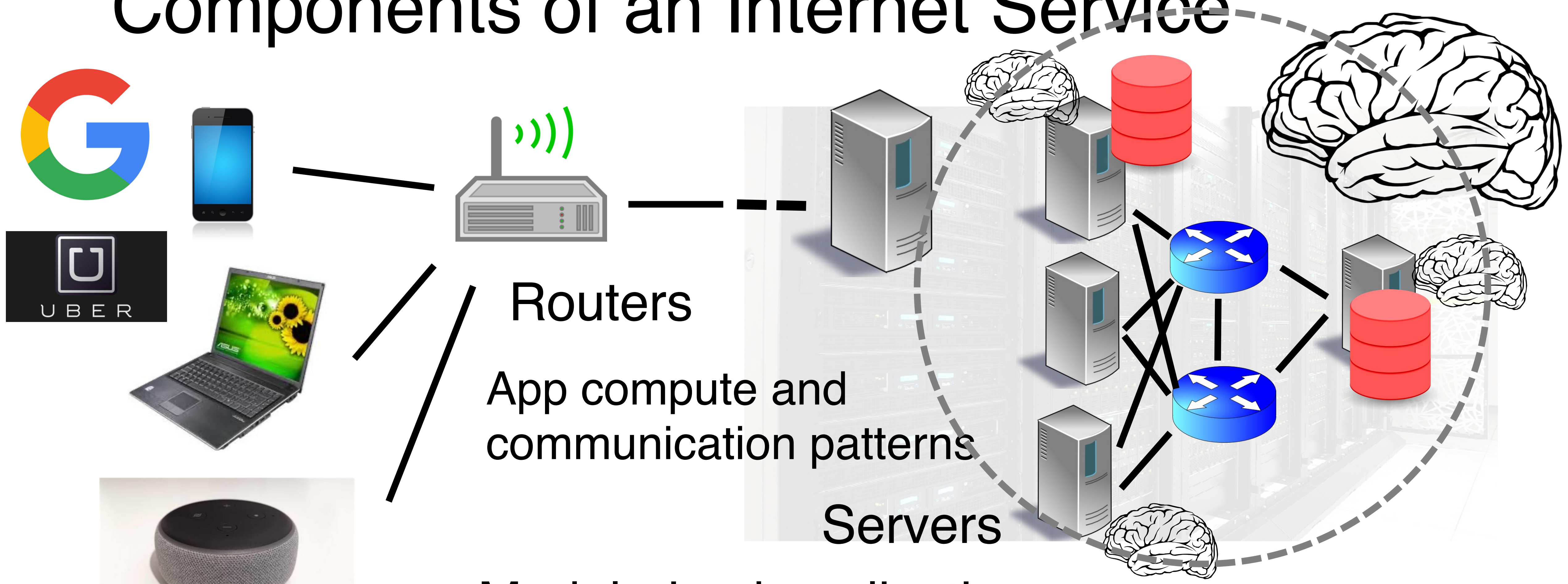
Components of an Internet Service



Endpoints

Data Center

Components of an Internet Service



App compute and communication patterns

Modularized applications

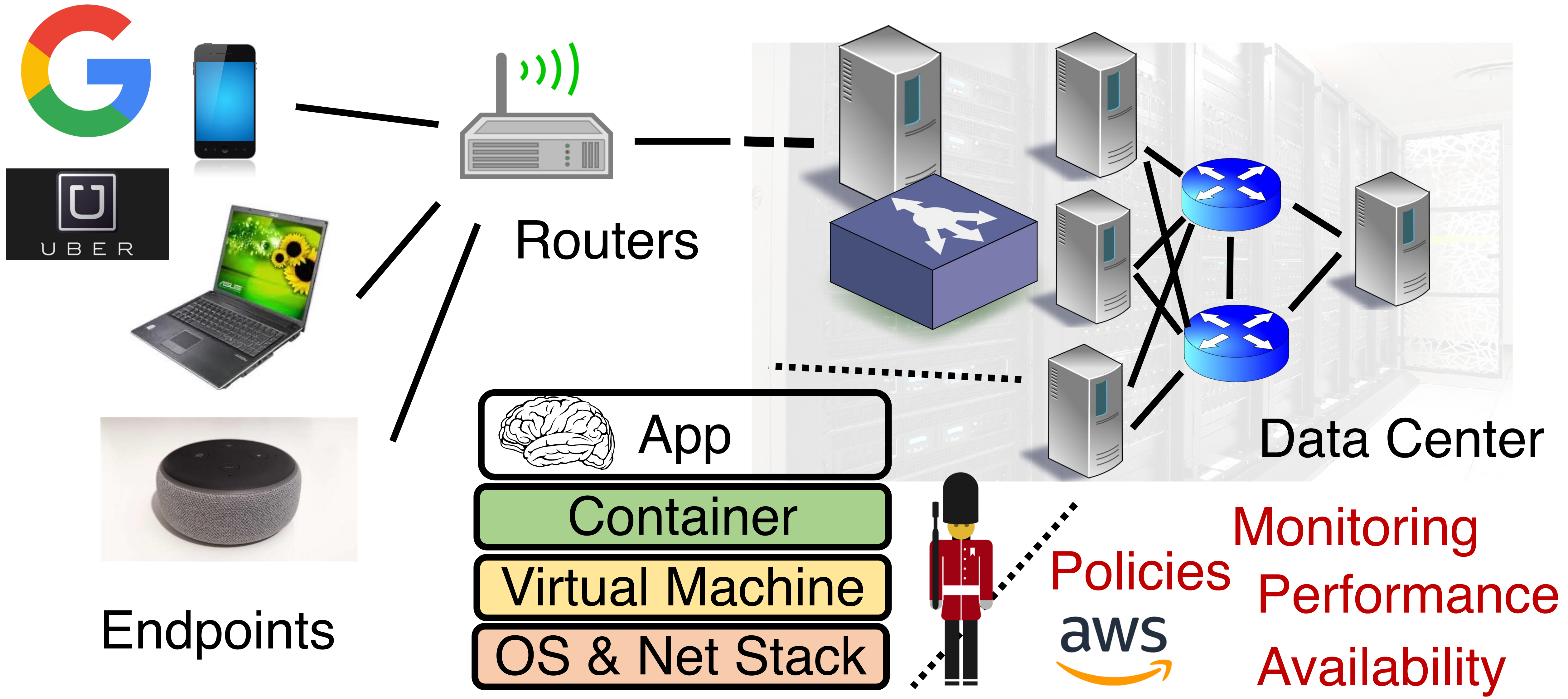
Data Center

Endpoints

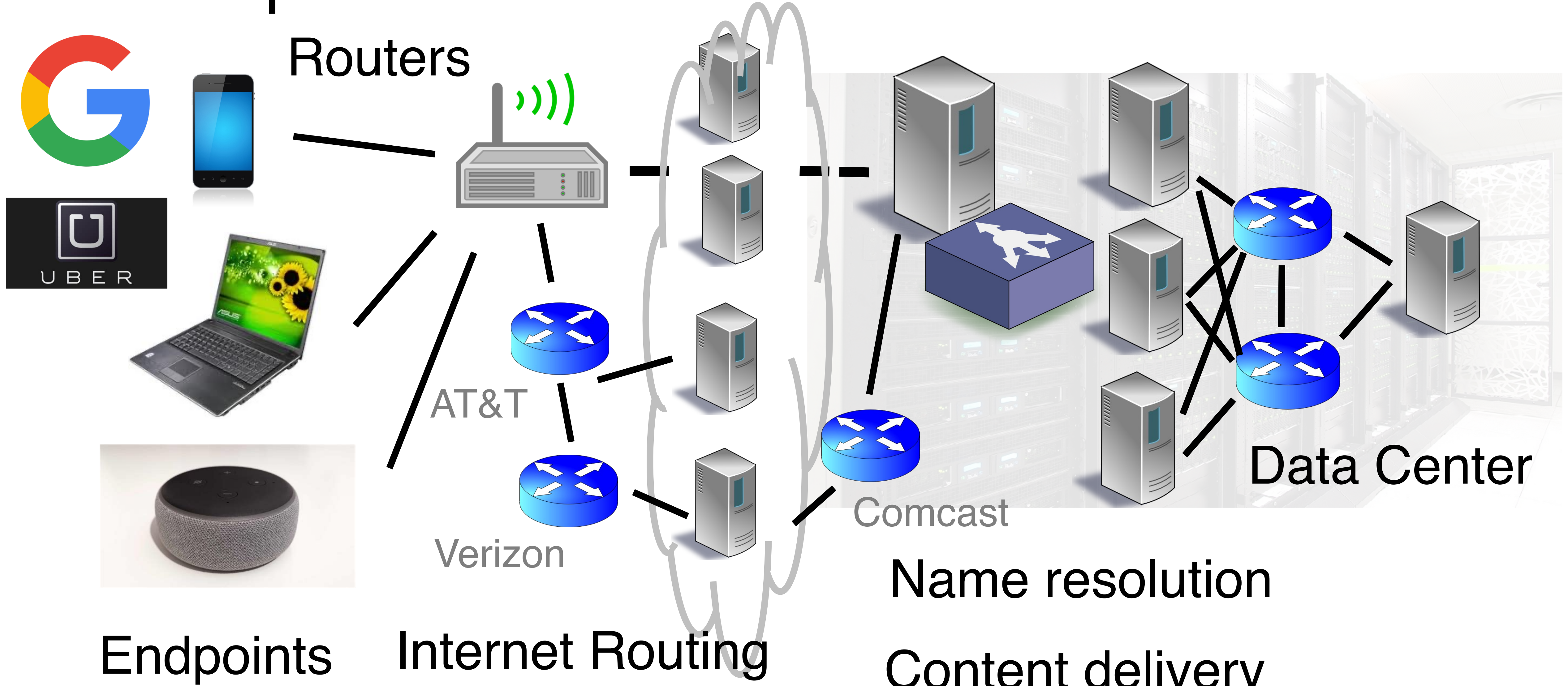
Storage

Interconnect: Routers

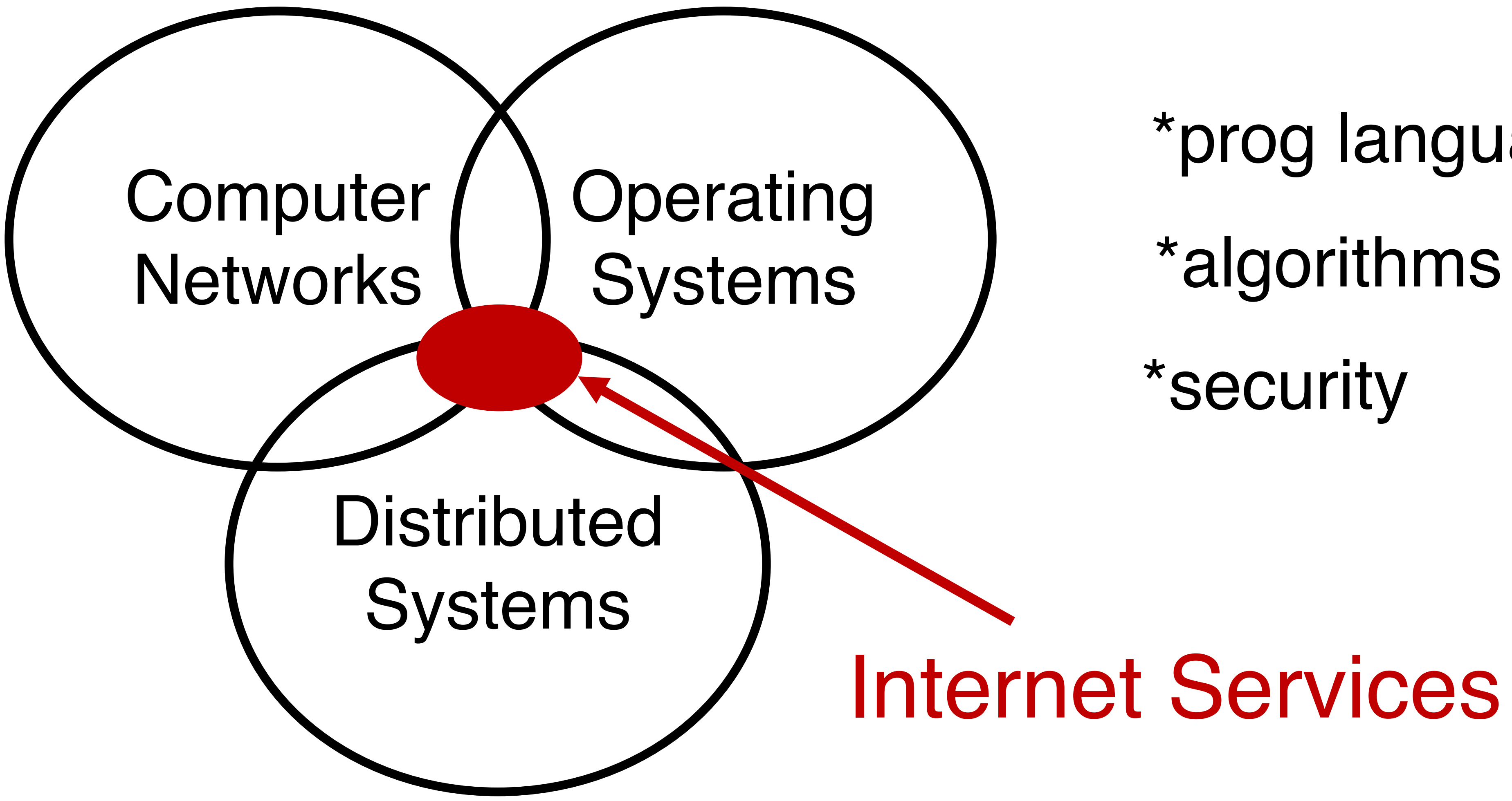
Components of an Internet Service



Components of an Internet Service



Core technical disciplines (incomplete)



*prog languages

*algorithms

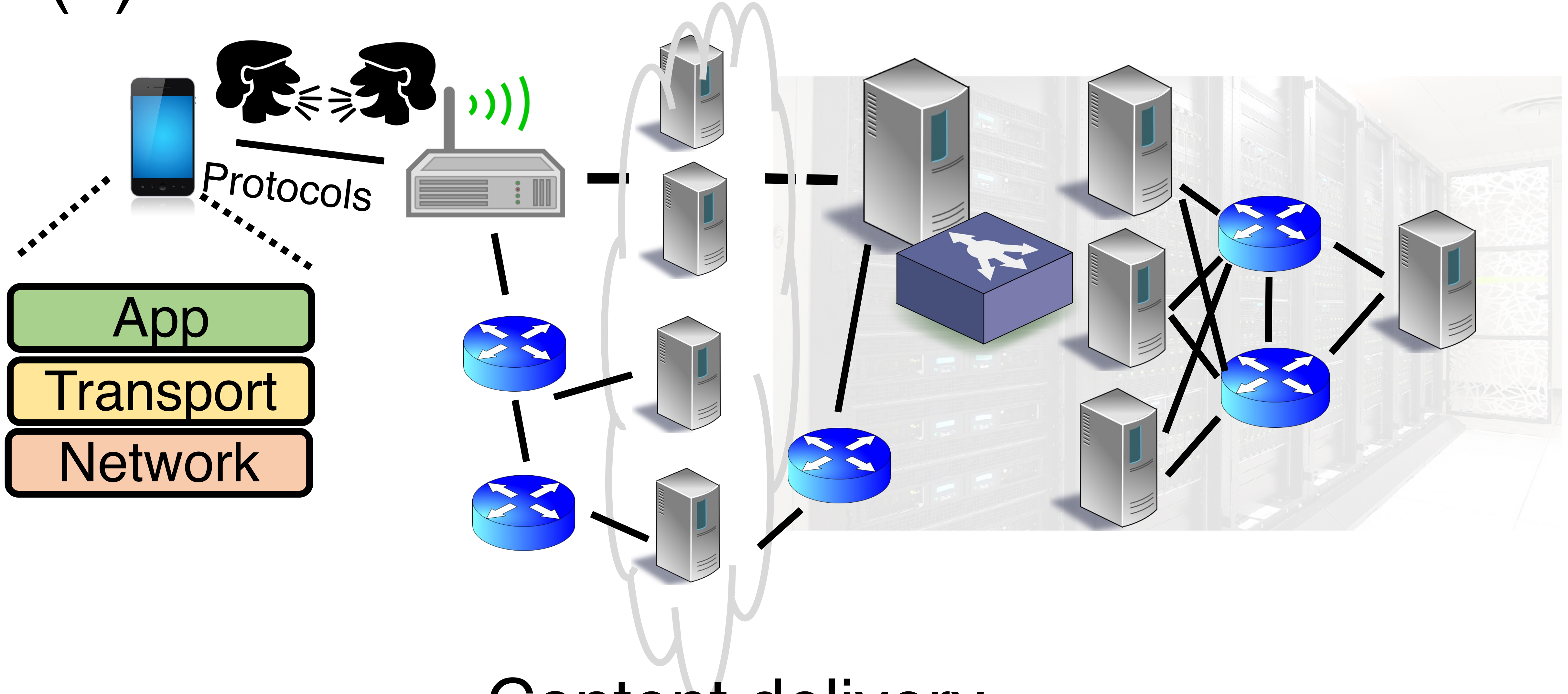
*security

Why should you study Internet services?

- Intellectual merits
  - Interdisciplinary problems, many principles
- Real world utility
  - Low barriers: anyone can do something useful
- Pragmatic
  - Many job opportunities, timely (stagnation of compute speed)
- Distinct from other coursework at Rutgers
  - OS, distributed systems, databases, networks

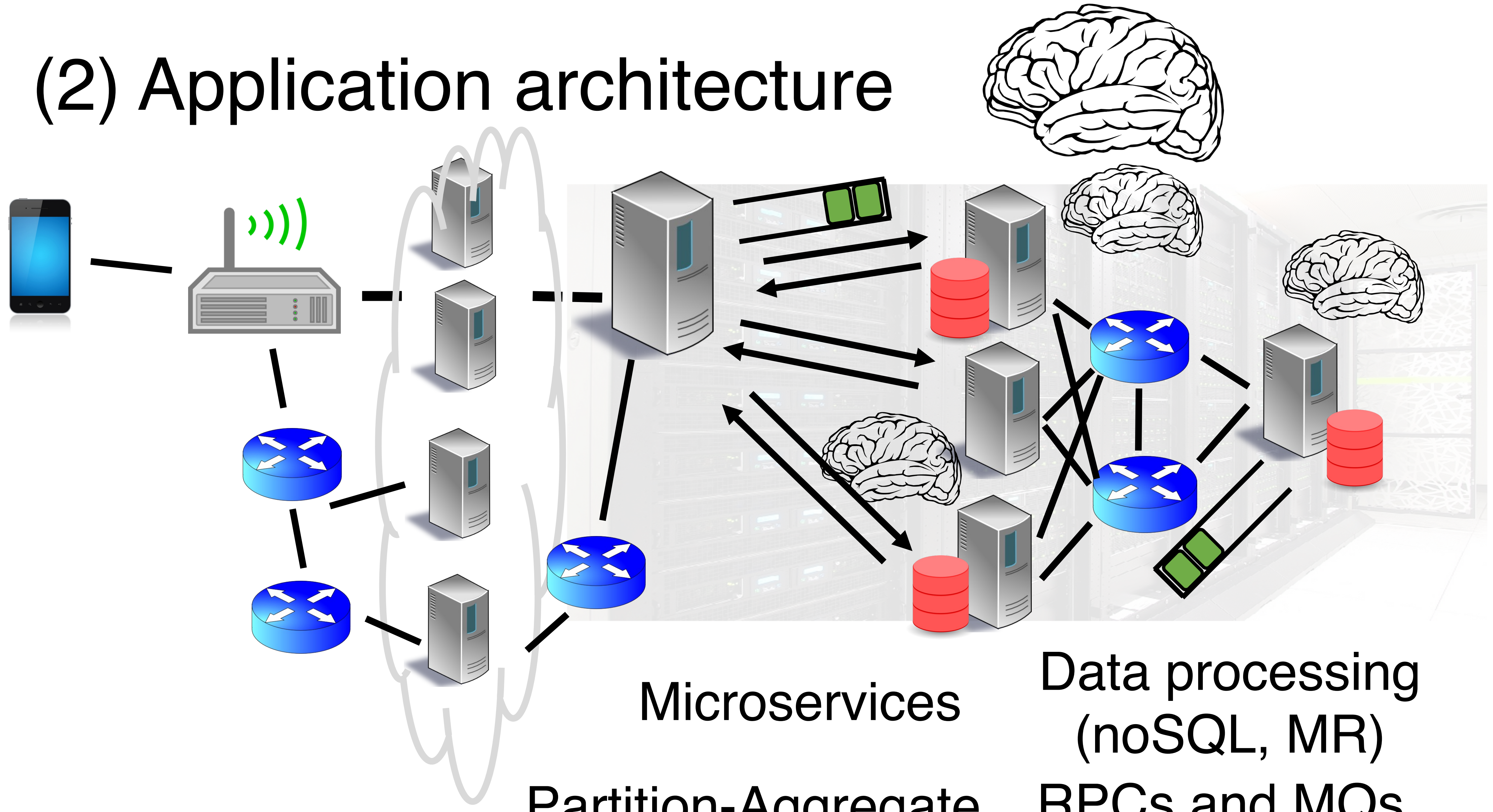
Course Content Overview

(1) Internet and Web Architecture



Content delivery

(2) Application architecture



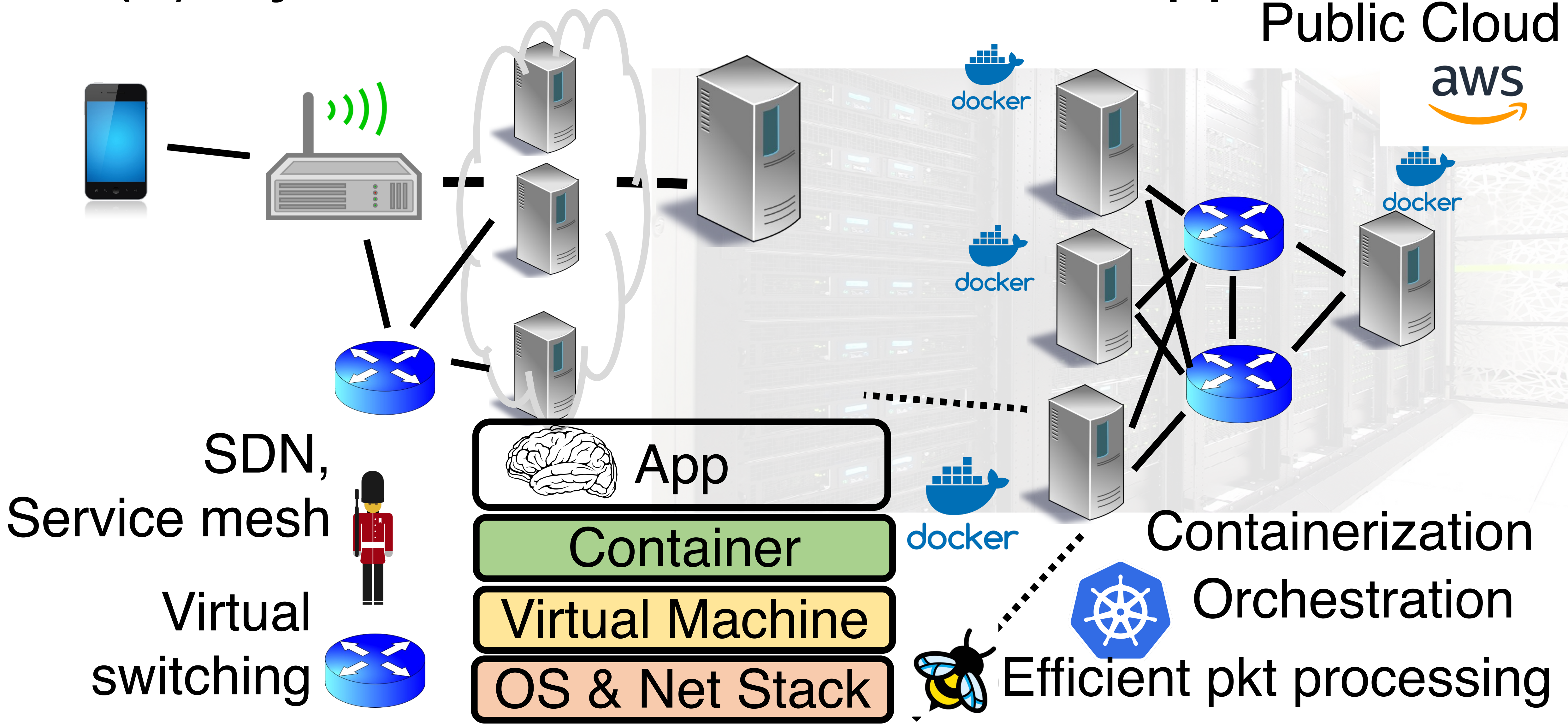
Microservices

Partition-Aggregate

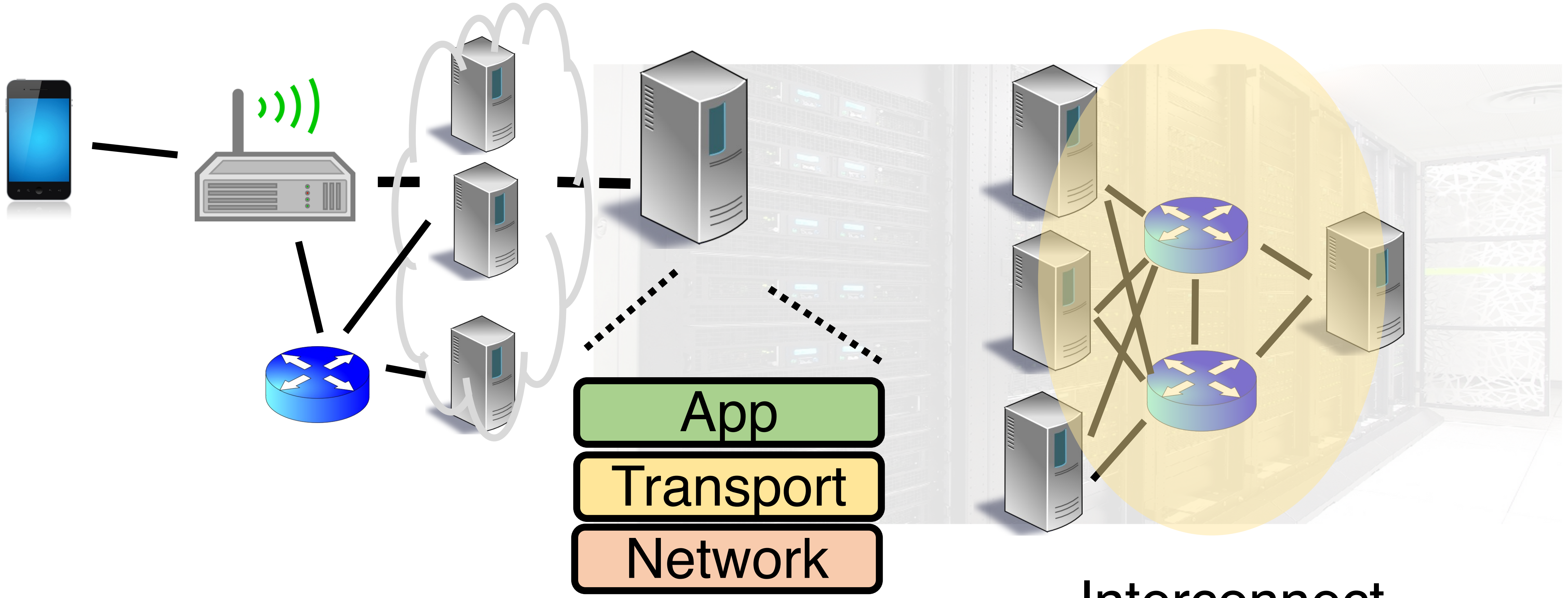
Data processing
(noSQL, MR)

RPCs and MQs

(3) System and Infrastructure support

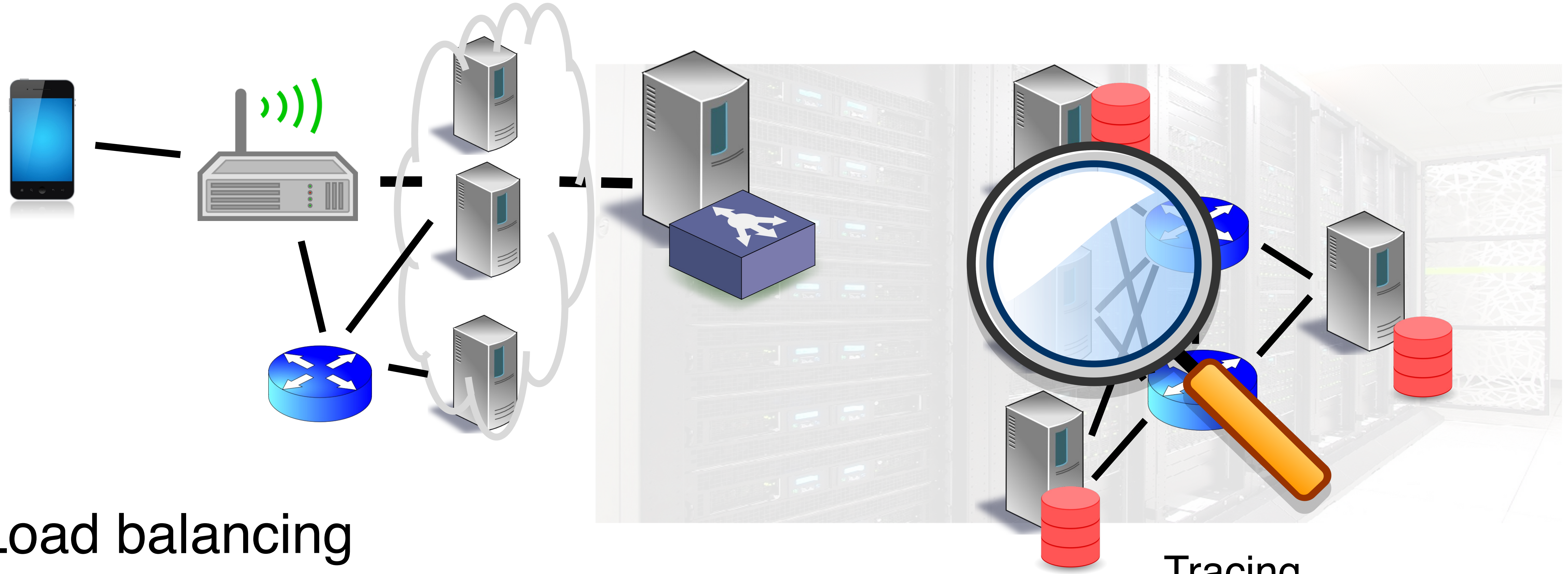


(4) Networking



Data center transport

(5) Ops: Availability, Perf, Monitoring



Load balancing

Fault tolerance, coordination services

Monitoring

Tracing

In-band network telemetry

Network stack monitoring

Course Logistics

About me

- Faculty Instructor: Srinivas Narayana
  - http://www.cs.rutgers.edu/~sn624
  - sn624@rutgers.edu
  - Office hours (student support hours) on Zoom: Mon 10 – 11 am ET and by appointment
  - Lecture Wed 8:30 – 11:30 in Busch SEC 118
- Canvas and Piazza
- Class info: http://www.cs.rutgers.edu/~sn624/553-S23/
  - Slides will be posted there

Class philosophy

- We want you to learn and to be successful

- Significant technical reading & programming
  - Build necessary skills for future tech careers (industry or academic)

- Be proactive: interact, ask, support
  - Attend lectures and office hours regularly to discuss material
  - Use Piazza

- Happy to help develop necessary background
  - Ask me for support materials

Course grade components

- Online quizzes (~20%)
- Programming homeworks (30%)
- Course project (~40%)
- Class participation (10%)
- Absolute grading; no curve

Online quizzes (~20%)

- Handed in on Canvas every Tuesday and Friday
- Answer questions on the lecture and assigned reading
  - Requires significant engagement with the technical readings
  - ~5 hours per reading
- One hour to complete a quiz once you start
- Due by 8 pm on the assigned date
- Only consider the ~20 highest grades (drop lowest ~5)

Online quizzes (~20%)

- Welcome to discuss readings and lectures with me and your peers

- However, quiz must be taken alone
  - Don’t reveal quiz questions or answers to other students

- No collaboration, googling, or generative AI (chatGPT)

- However, you can consult the lecture and the technical paper while taking the quiz
  - Open book but not “open Internet”

You’ll spend significant time reading

- This course has 2 assigned technical readings per week
    - Technical blog posts, accessible survey articles, deep technical papers
- Knowing how to read well technically is worth your time
    - Grad students: reading and writing technical papers
    - Developers: reading RFCs, protocol specifications, other technical specs
    - Development: implementing new system technology requires reading
- Staying broadly technically educated (and employable!)
- It’s worth reflecting on how to read effectively

There is no magic

- Reading effectively takes a lot of time and effort
- I’ve been reading technical articles for 12+ years now
    - And I still sometimes spend 5+ hours or even an entire day
- You will get more effective and better over time
- A few tricks

(1) Three pass approach for papers

- First pass: title, category, context, assumptions, correctness, contribution
- Second pass: get into the technical ideas, figures and graphs. Explain the new technical idea or design to someone else
- Third pass: starting from assumptions and problem statement, reconstruct solution on your own with proof or argument for why it works and why it is the right approach (other approaches?)

How to read a paper? -- Keshav Srinivasan

(2) Look for concrete, simple examples

- Good articles often use good examples to explain their ideas
- If the article does not show examples, construct and work through your own
- Constructing a good example itself often clarifies the technical problems and innovations in the solution

(3) Identify reusable principles

- What do you want to take away from the reading?

- System design or algorithmic techniques
- Existence of a new problem space
- A class of technical solution approaches
- New techniques for empirical evaluation or measurement

- Reading something worthy changes how you think about the world from that point

Programming homeworks (30%)

- Four over the first half of semester, by yourself
  - Tentatively due 2/06, 2/20, 3/06, and 3/27
- Released and handed in on Canvas roughly every two weeks
- C/C++ programming language, Python, shell
- Test and run on a Linux VM (instructions to be provided)
- Some of the projects may need superuser privileges

Programming homeworks (30%)

- Please follow all instructions carefully and exactly

- You will lose significant points if:
  - I am unable to run your code
  - I do not receive your submission in a timely fashion

Programming homeworks (30%)

- You can collaborate with others freely, however all submitted code must be your own work
- Do not blindly lift code from stack overflow, GitHub, chatGPT, etc.
- Incorporate learning from other sources and produce your own solutions
- Mandatory to state collaboration & references at the beginning of submitted program

Course project (~40%)

- Team of 2, latter half of the semester
  - Please, no larger teams or individual work
- Aligned with the course topics
- “Significant” programming component
  - Measured by complexity and the size of the source code
  - Talk to me to ensure (“project specification” info coming up)

Course project (~40%)

- A small open-ended research problem
- Adding new features to an existing open-source codebase
- Reproducing empirical evaluations and benchmarks from a paper you read
- Re-implementing an existing technique on a different system
- Building a tool that makes further technical work or research possible or easier

Course project (~40%)

- I will send out some possible ideas
- Welcome & encouraged to work on something *you* find exciting
- Form teams, brainstorm ideas with each other and me early
- I will help you succeed technically: don’t struggle alone
- 3 concrete deliverables: project specification, source code, technical report

Project specification

- 1—2 pages, tentatively due 4/03

- Specific, measurable, realistic technical goal
- Existing prior work and why your goal or approach is novel
- Brief description of technical idea and solution approach
  - What needs to be built? How will you build it?
- Key performance metrics, qualitative and quantitative, you will evaluate your system on
- Technical and other risks

Source code

- Due end of the semester (tentatively 5/04)
- Host source code on public repository e.g., on GitHub
- Provide clear documentation (README) with commands and requirements to run your system
- Provide scripts and commands to reproduce your empirical evaluation results

Technical report

- 5—10 pages, due end of the semester (tentatively 5/04)
  - Latex template to be provided
  - One PDF per team submitted on Canvas
- Clearly and fully describe all the technical details:
- Implementation of the solution
- How you evaluated the system: metrics, workloads, executions
- Why did you observe the numerical results that you did?
- Optional: presentations or demonstrations of the system
- Assessed for clarity, comprehensiveness, technical design, scientific accuracy

Course project (~40%)

- Don’t “just implement” something, also measure and explain it

- Highly coveted technical skills:
  - Identifying good performance indicators, representative workloads an configurations
  - Measuring them accurately
  - Explaining them clearly with more detailed measurements

- Rigorously evaluating a system may take as long as implementing it

Course project (~40%)

- Do not blindly lift code from other sources
- When you use existing software libraries, state the nature and scope of their use clearly in your project spec and report
- Do not blindly lift text (e.g., for project report) from other sources
- Please cite references for specific statements and be thorough

Course participation (10%)

- You are welcome to discuss and collaborate extensively
    - Get to know each other, and me

- Meaningful class questions and technical discussion
- Insightful piazza questions or answers or follow-up discussion
- Discussion with me after lecture or in office hours
- Supporting and helping each other grow in any way
    - e.g. sharing useful materials through Piazza

Course participation (10%)

- Class participation is a consistent and meaningful activity
  - Assessed throughout the semester
  - Not a one-time event or a checkbox
  - Intention to learn and support, not just a grade
- I’m happy to get to know you professionally and engage in technical discussions
  - If you require professional support from me later (e.g., recommendation letters), it’s a great way for me to know you better

Collaboration and Integrity policies

- This course welcomes discussion and collaboration
- Do
  - Ask questions on Piazza
  - Discuss projects and readings with me and with each other
  - Read references (textbooks, papers, Internet posts) widely
  - Acknowledge each other and all the references
- Use collaboration prompts on programming homeworks; project
  - Include who you talked to, references (including on the web) you consulted
  - Be as accurate and complete as possible

Collaboration and Integrity policies

- All your written (coded) work must be your (team's) own
  - Understand the problem deeply and produce your own solutions
- Do not
  - blindly lift or incorporate other solutions
  - look at other people's code or solutions
  - copy code from the web (e.g., other people's GitHub projects)
  - use generative AI (e.g. chatGPT)
  - post programming homeworks or quizzes (questions or solutions) on GitHub, Chegg, CourseHero, etc.

Rutgers takes academic dishonesty very seriously.

Violation of academic integrity at the graduate level is especially serious. Consequences include suspension and expulsion.

We will run plagiarism detection tools on all submitted materials.

If you are ever in doubt, ask me first.

Late policy

- Don’t be late
- If you must be late, inform us in advance
- If you cannot inform us in advance (e.g., medical), provide official medical note of absence through the University
- Unexcused late submissions will result in losing significant fraction of points

24/7 Grading Policy

- You may not dispute a grade or request a regrade before 24 hours or after 7 days of receiving it

- Please contact us if you have a legitimate regrading request:
  - After 24 hours of receiving the grade: Please take the time to review your case before contacting me
  - Before 7 days have elapsed: we don’t want to forget what the quiz/project was all about.

Help, Accommodations, etc.

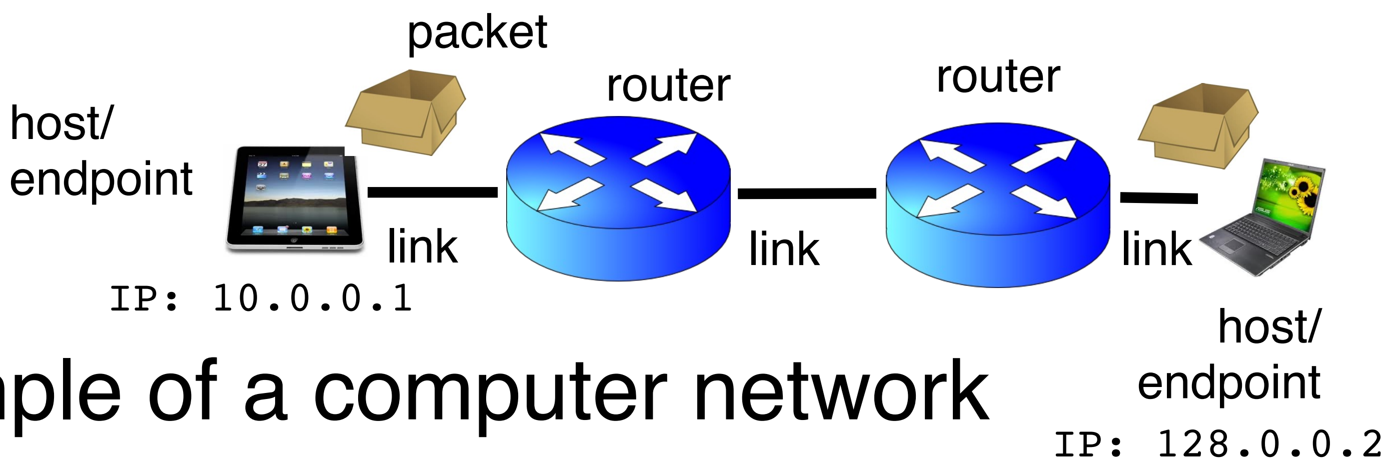
- I’ll make every effort to accommodate reasonable requests that support your learning better
- sn624@cs.rutgers.edu
- I am committed to help you succeed in this course.

Next steps

- Sign up for class Piazza: link on canvas home page
- Warm up on C/C++ programming this week
    - e.g., linked lists, basic TCP and UDP socket programming
- First programming homework released on Monday
- First quiz due next Tuesday (announcement)
    - Thereafter due every Friday and Tuesday
- Meet each other and form project teams

Internet Architecture

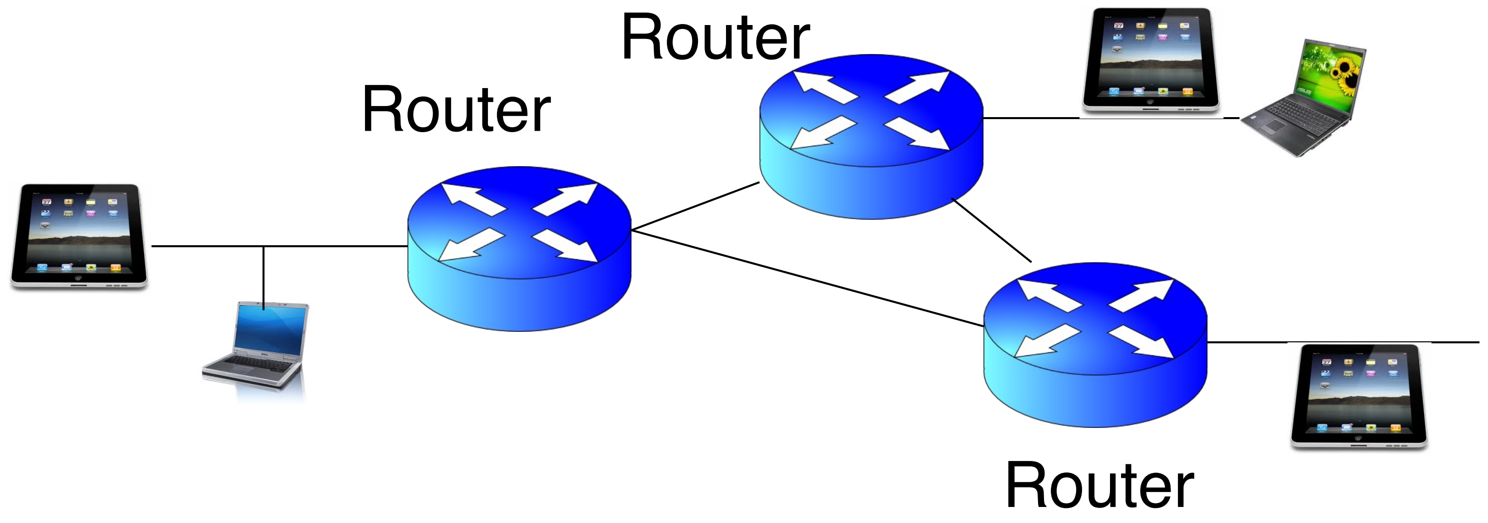
Some definitions



- The Internet is an example of a computer network
- Endpoint or Host: Machine running user application
- Packet: a unit of data transmission (ex: 1500 bytes)
- Link: physical communication channel between two or more machines
- Router: A machine that processes packets moving them from one link to another towards a destination
- Network: Collection of interconnected machines
- Address: a unique name given to a machine

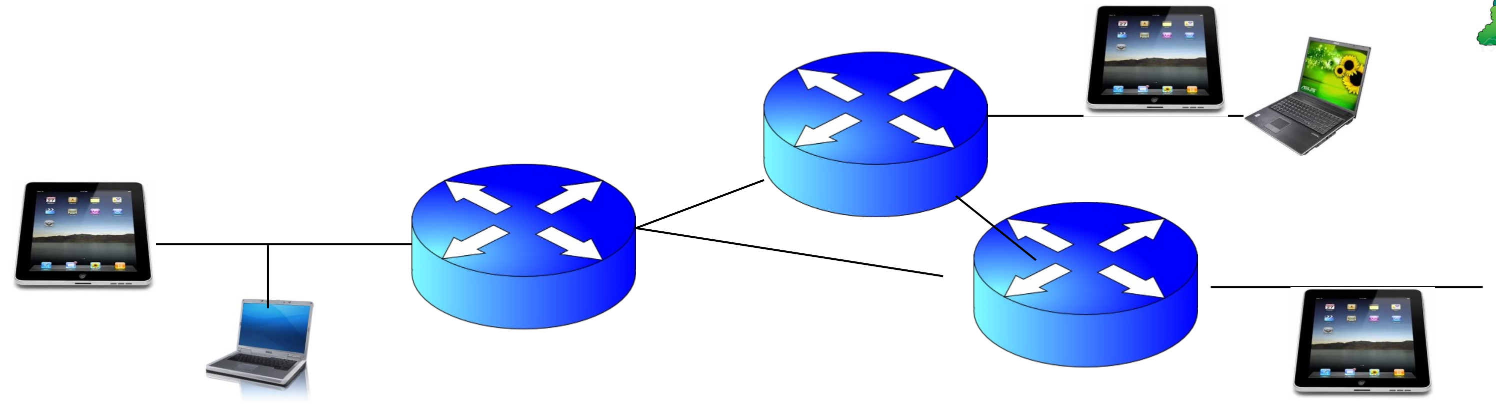
Some fundamental problems

(1) Routing



- Networks must move data between different hosts
- Need to figure out how to move packets from one host to another host, e.g., how to reach google.com from your laptop
- Known as the routing problem

(2) Name Resolution



- Routing effectively requires locating the endpoints appropriately
  - Memory, speed, reactivity
- Internet addresses allocated hierarchically
  - Machine readable, not easy for humans to remember
- Link addresses are tied to the hardware on the endpoint
- Name resolution: how to turn human-readable names (google.com) into routable addresses?

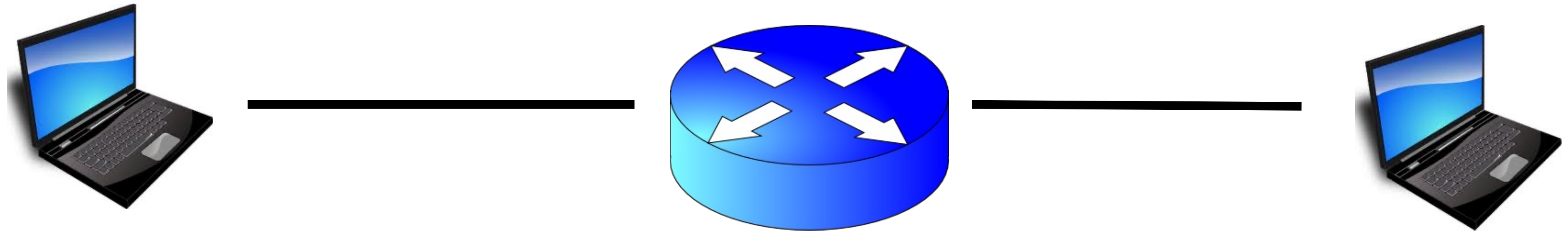
In general, networks give no guarantees

- Packets may be lost, corrupted, reordered, on the way to the destination
  - Best effort delivery

- Advantage: The network becomes very simple to build
  - Don't have to make it reliable
  - Don't need to implement any performance guarantees
  - Don't need to maintain packet ordering
  - Almost any medium can deliver individual packets
    - Example: RFC 1149: "IP Datagrams over Avian Carriers"
- Early Internet thrived: easy to engineer, no guarantees to worry about

Providing guarantees for applications

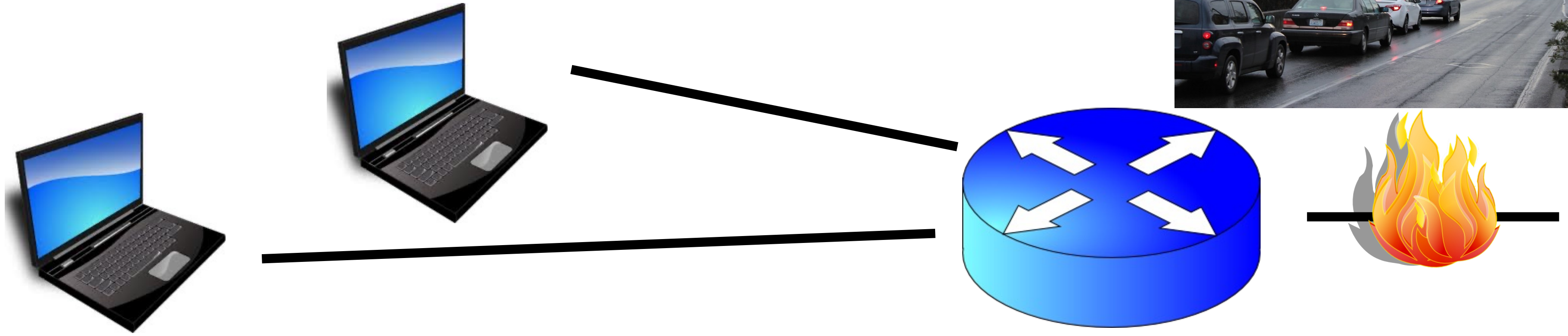
- How should endpoints provide guarantees to applications?



- Transport software on the endpoint oversees implementing guarantees on top of an unreliable network
- Reliable delivery, ordered delivery, fair sharing of resources

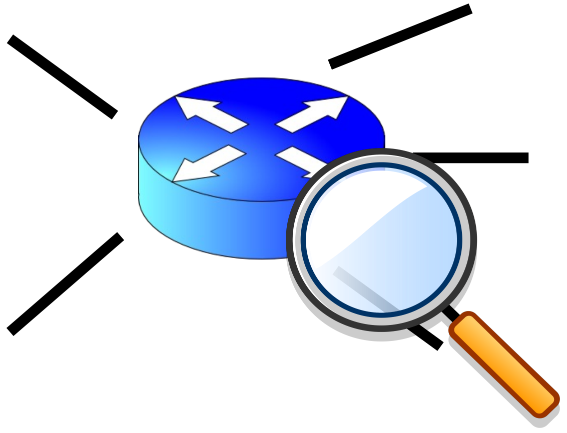
(3) Congestion control

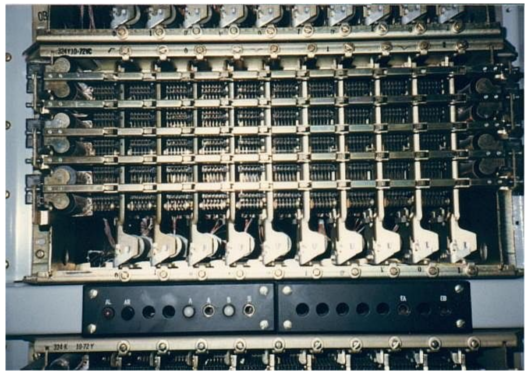
- How quickly should endpoints send data?

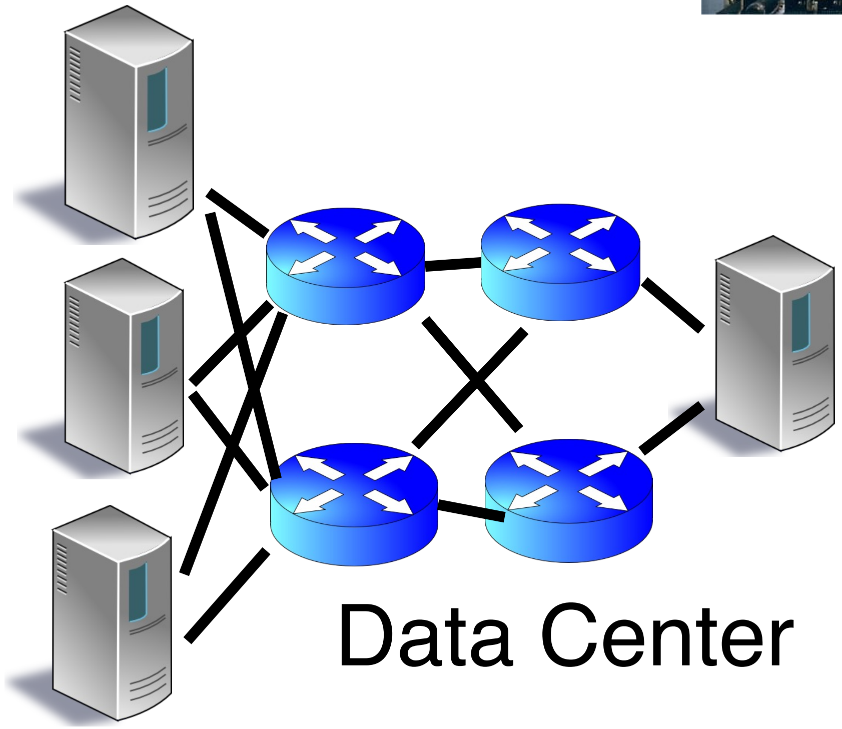


- Known as the congestion control problem
- Congestion control algorithms at source endpoints react to remote network congestion. Part of the transport sw/hw stack.
- Key question: How to vary the sending rate based on network signals?

(4) High-Speed Interconnect



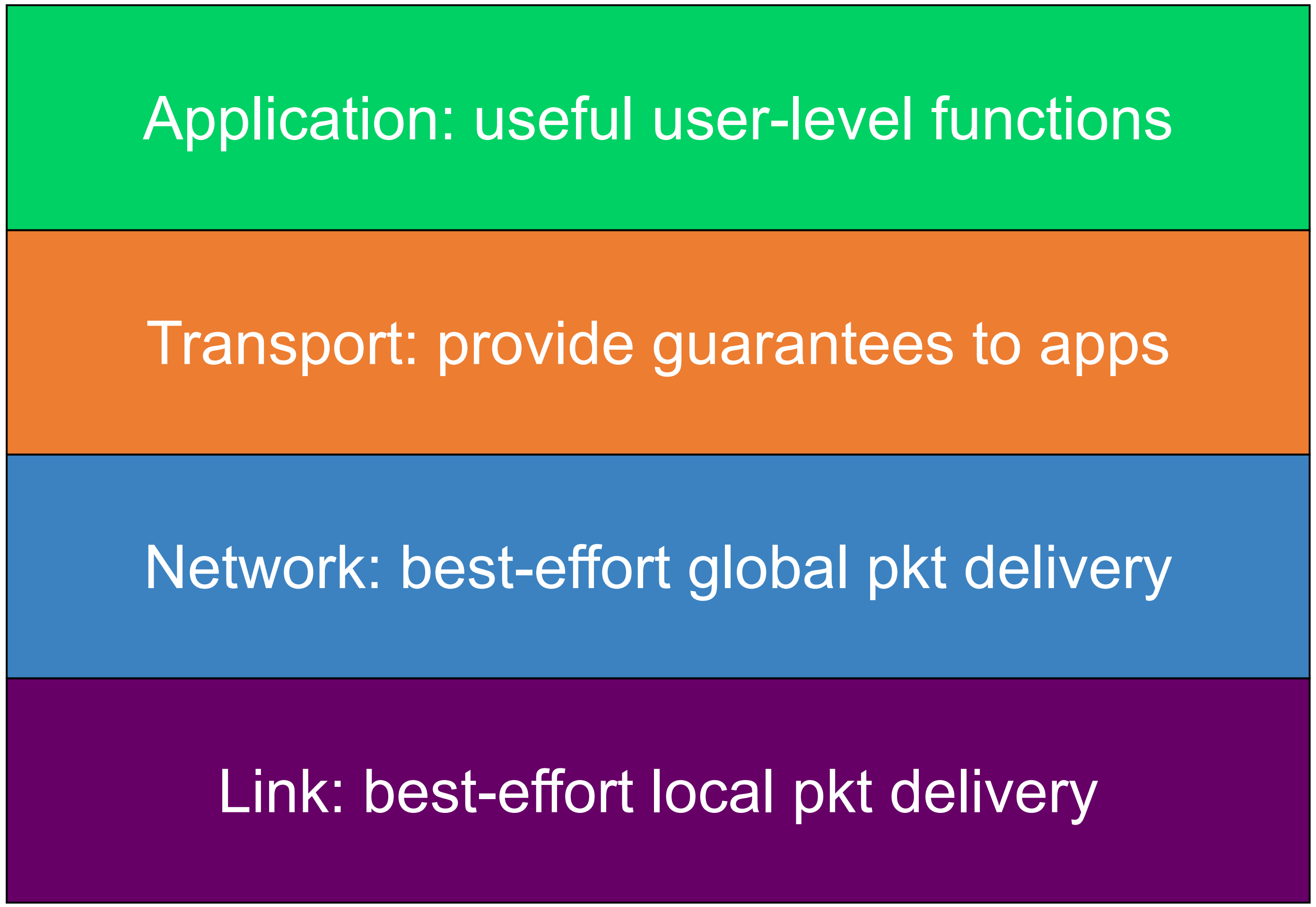




- Transport won't help if the network has choke points: e.g., routers
- The interconnection problem: how do you design routers to achieve high end-to-end performance between endpoints?
  - Also designing large data center networks

Layering and Protocols

Software/hardware organization at hosts


Application: useful user-level functions
Transport: provide guarantees to apps
Network: best-effort global pkt delivery
Link: best-effort local pkt delivery

Communication functions broken up and "stacked"

Each layer depends on the one below it.

Each layer supports the one above it.

The interfaces between layers are well-defined and standardized.

Internet software and hardware are arranged in layers.

Layering provides modularity

Each layer: well-defined function & interfaces to layers above & below it.

Functionality is implemented in protocols.

Protocols: The “rules” of networking

- Protocols consist of two things

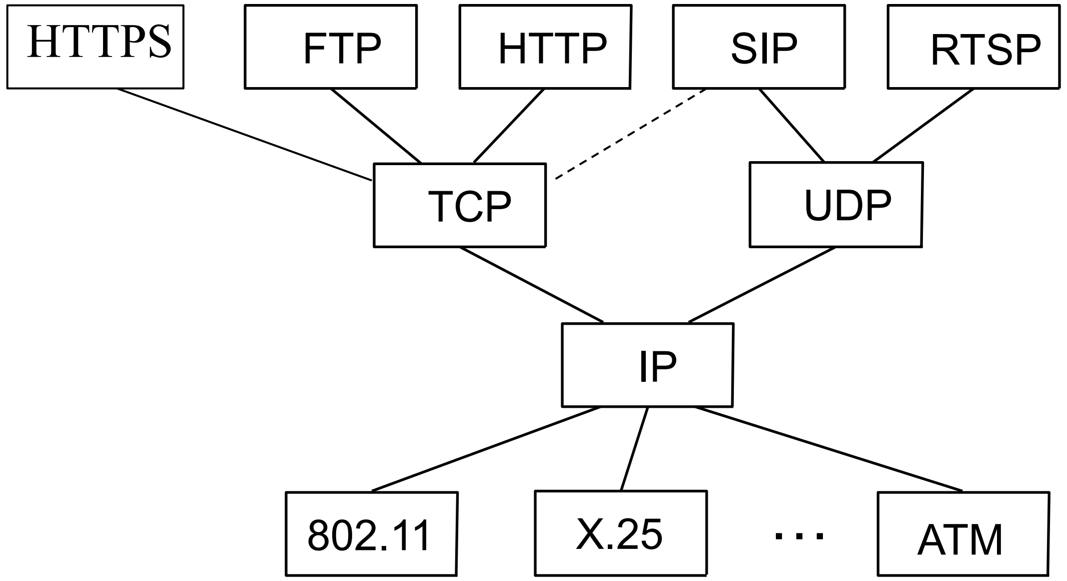
- Message format
  - structure of messages exchanged with an endpoint

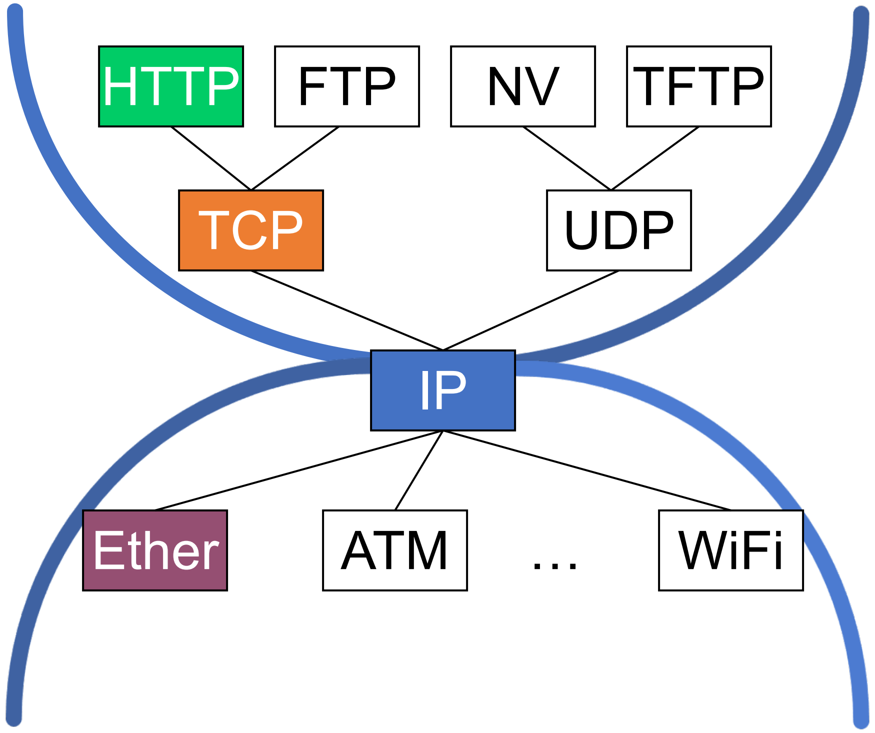
- Actions
  - operations upon receiving, or not receiving, messages

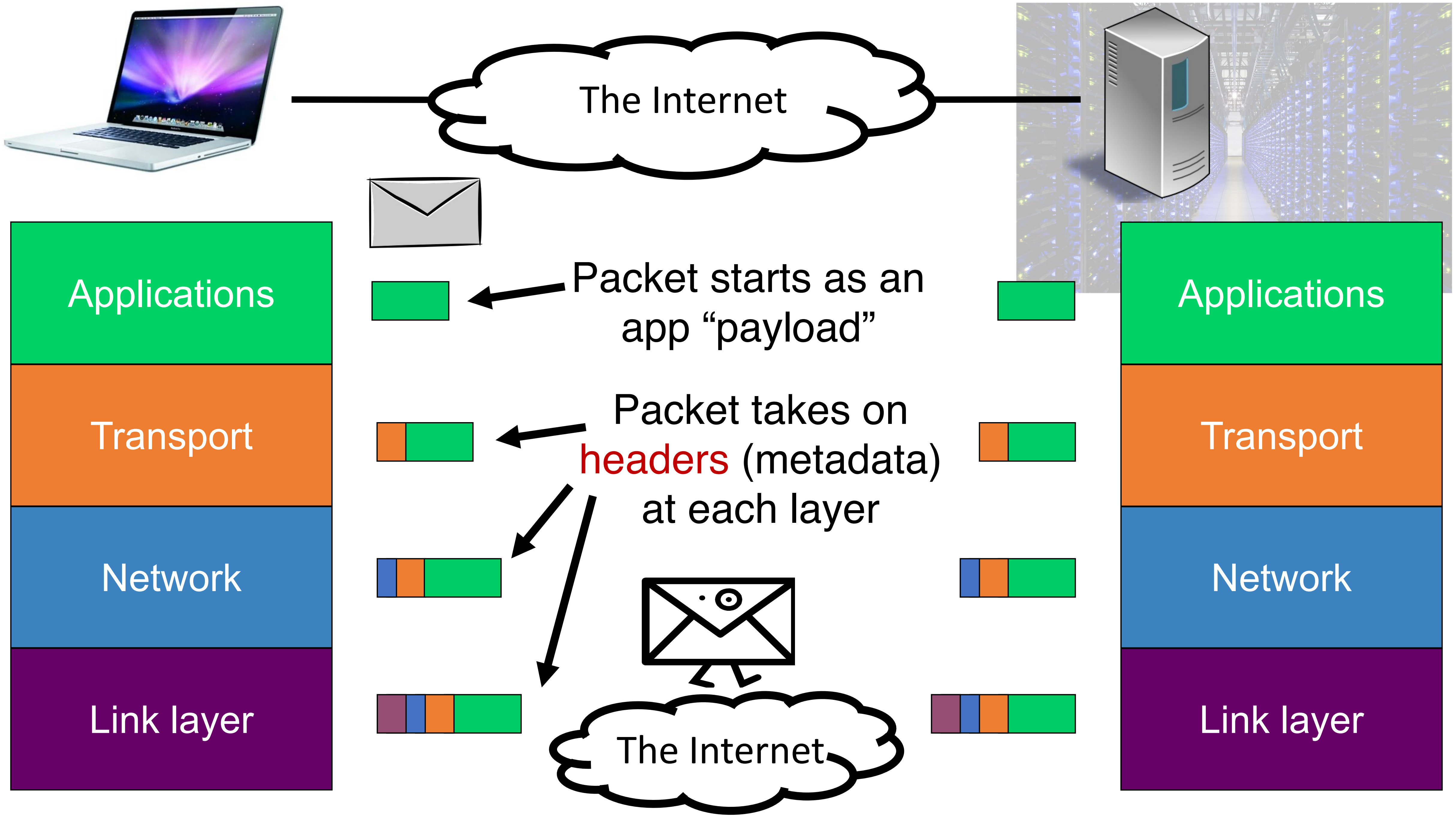
- Example of a Zoom conversation:
  - Message format: English words and sentences
  - Actions: when a word is heard, say “yes”; when nothing is heard for more than 3 seconds, say “can you hear me?”

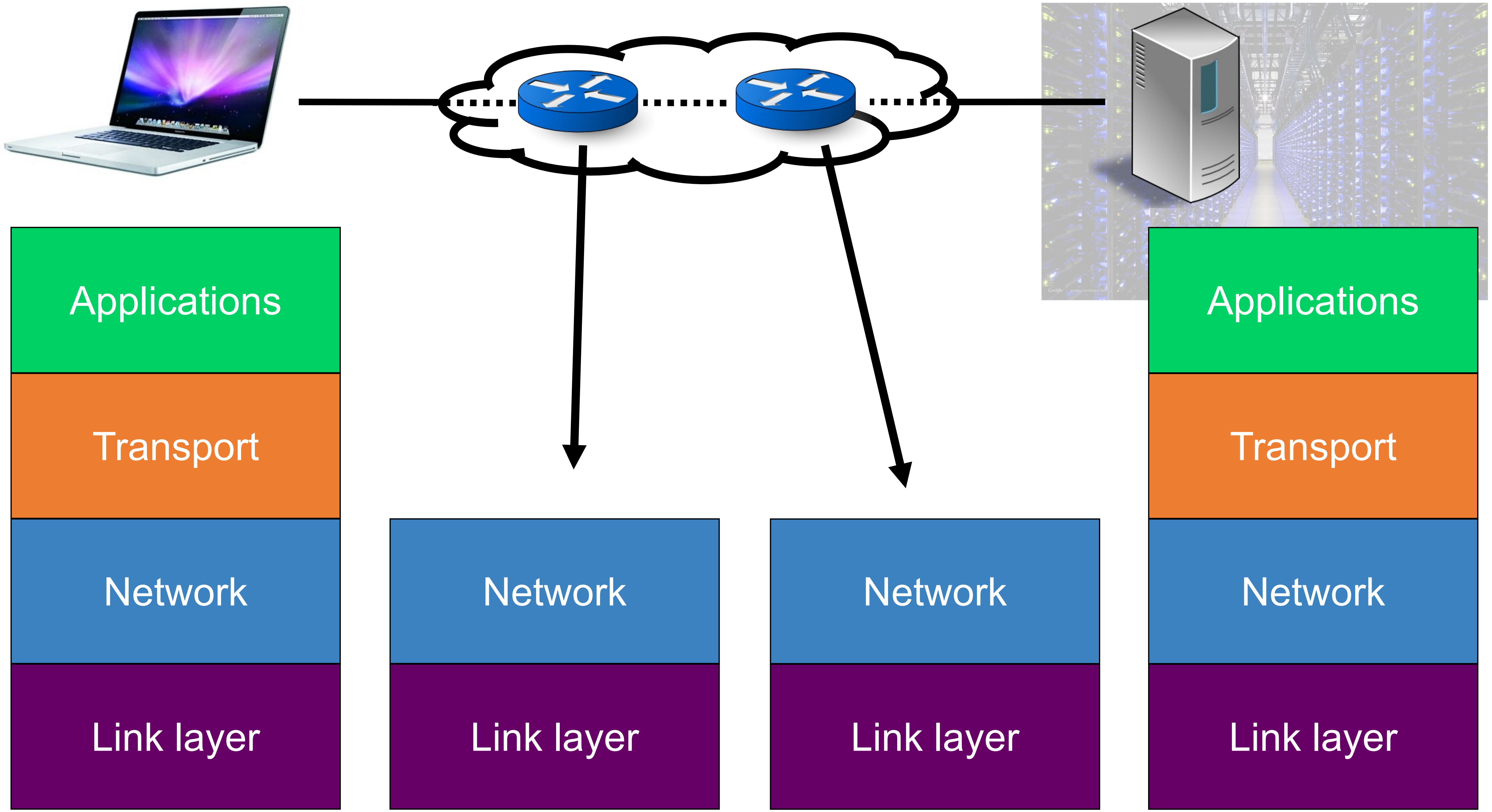
The protocols of the Internet

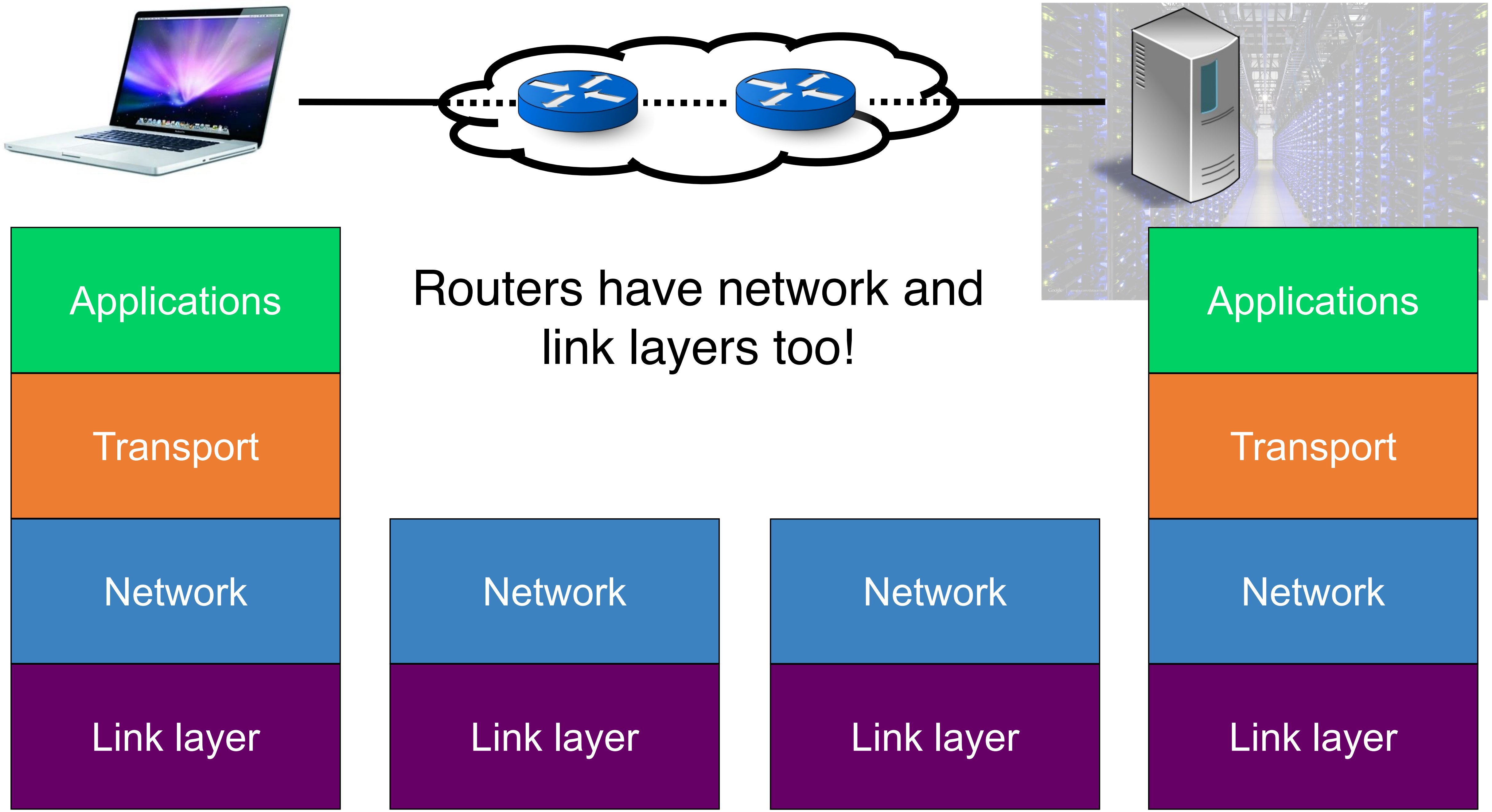
- Standardized by the Internet Engineering Task Force (IETF)
  - through documents called RFCs ("Request For Comments")
- Layering of protocols






The Internet
Packet starts as an app "payload"
Packet takes on headers (metadata) at each layer
Applications
Transport
Network
Link layer
Applications
Transport
Network
Link layer
The Internet


Applications
Transport
Network
Link layer
Network
Link layer
Network
Link layer
Applications
Transport
Network
Link layer



Routers have network and link layers too!

Layering

- Communication over the Internet is a complex problem.
- Layering simplifies understanding, testing, maintaining
- Easy to improve or replace protocol at one layer without affecting others