Service Delivery Architecture

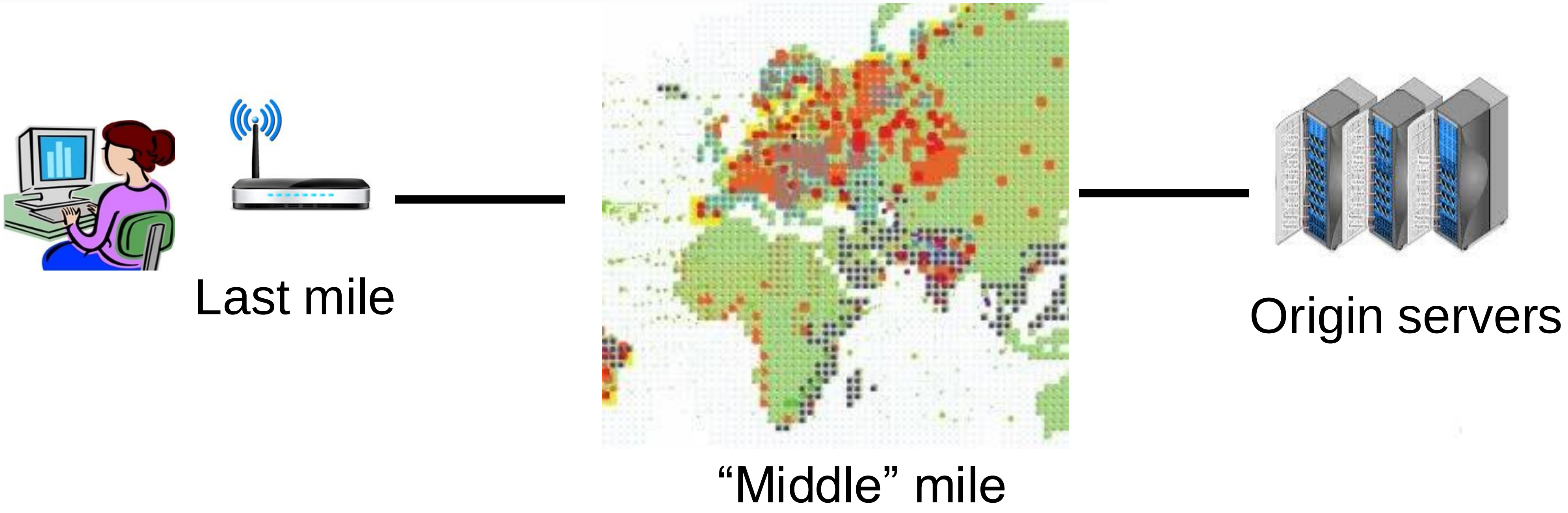
Lecture 4a

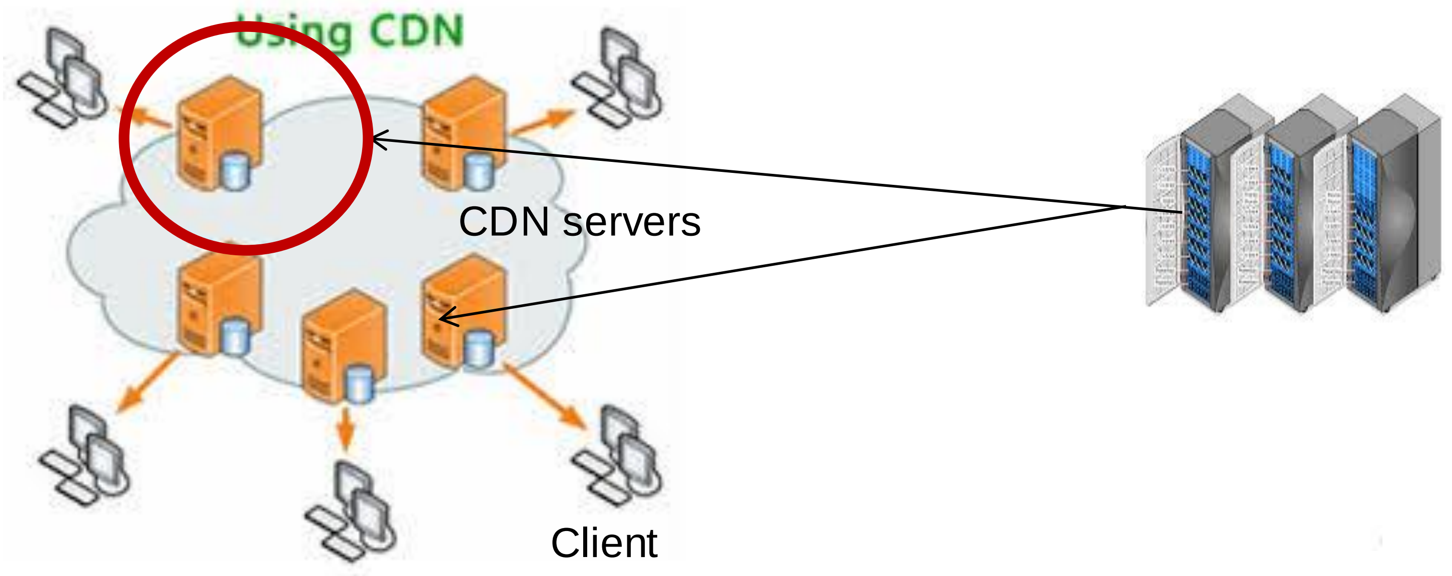
Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S25

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Improving Performance on the Internet





Internet Routing

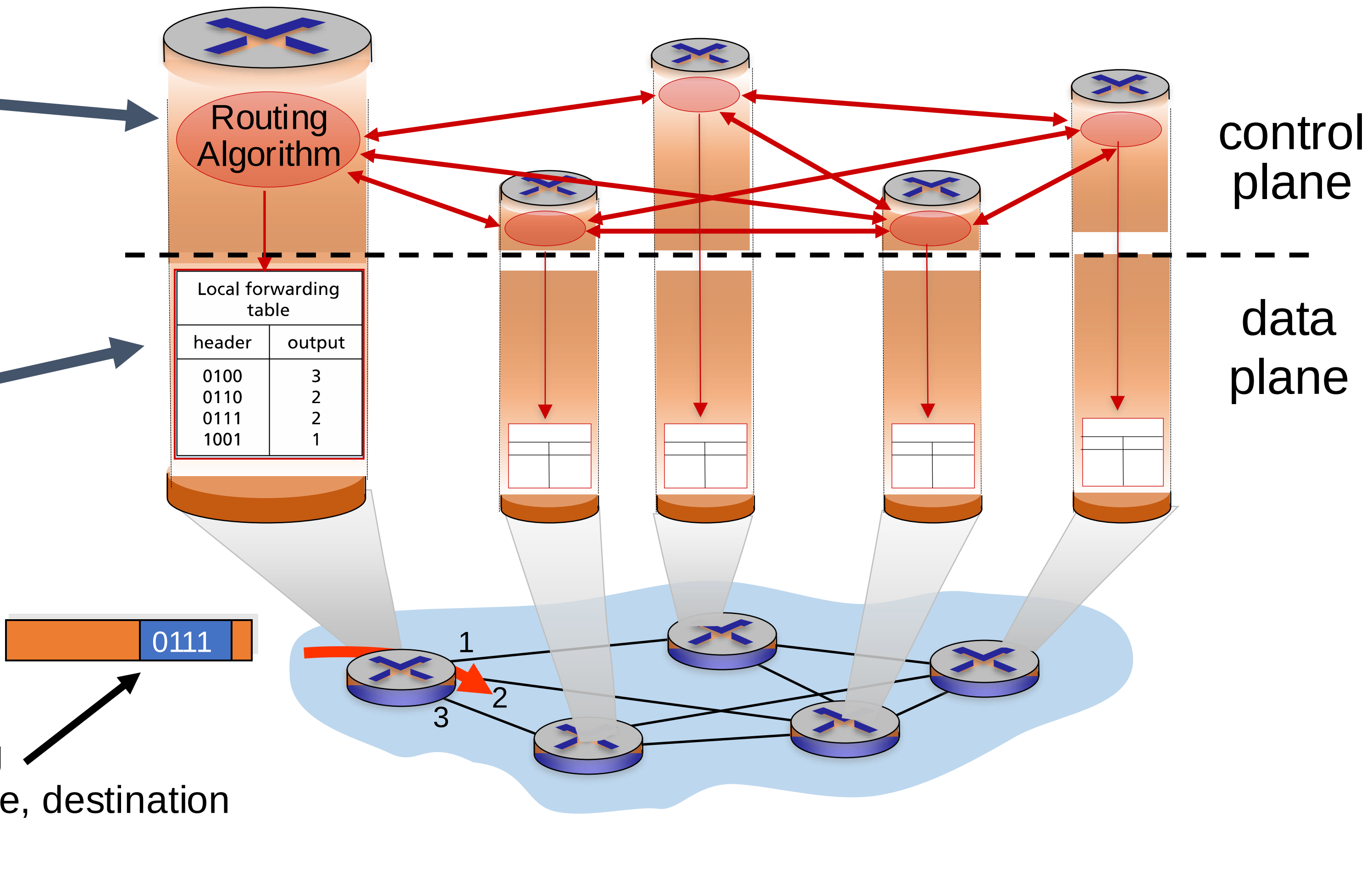
Distributed routing

Control plane

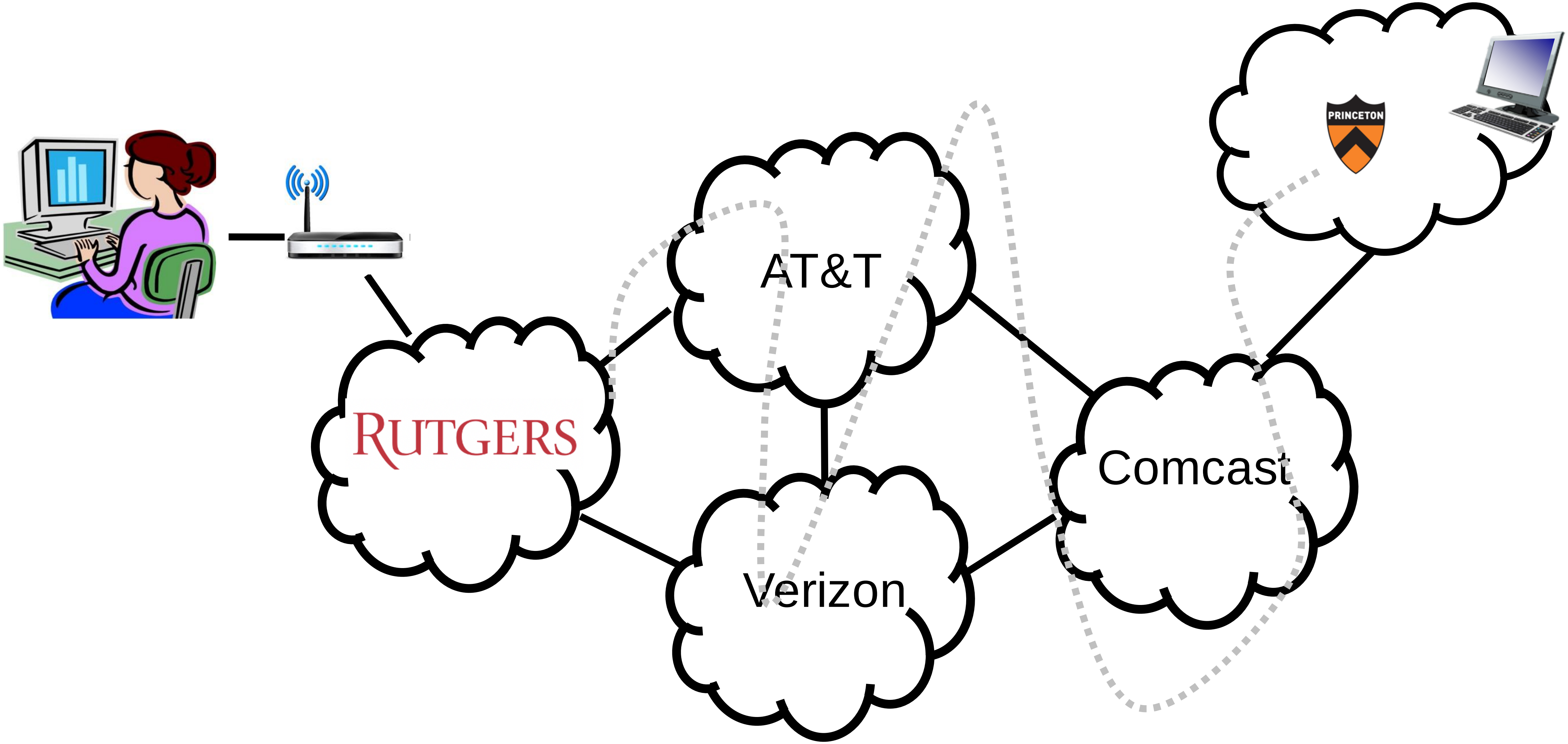
Traditional routing protocols: per route-change processing (~ a few tens of seconds)

Data plane

per-packet processing (~ tens of nanoseconds)



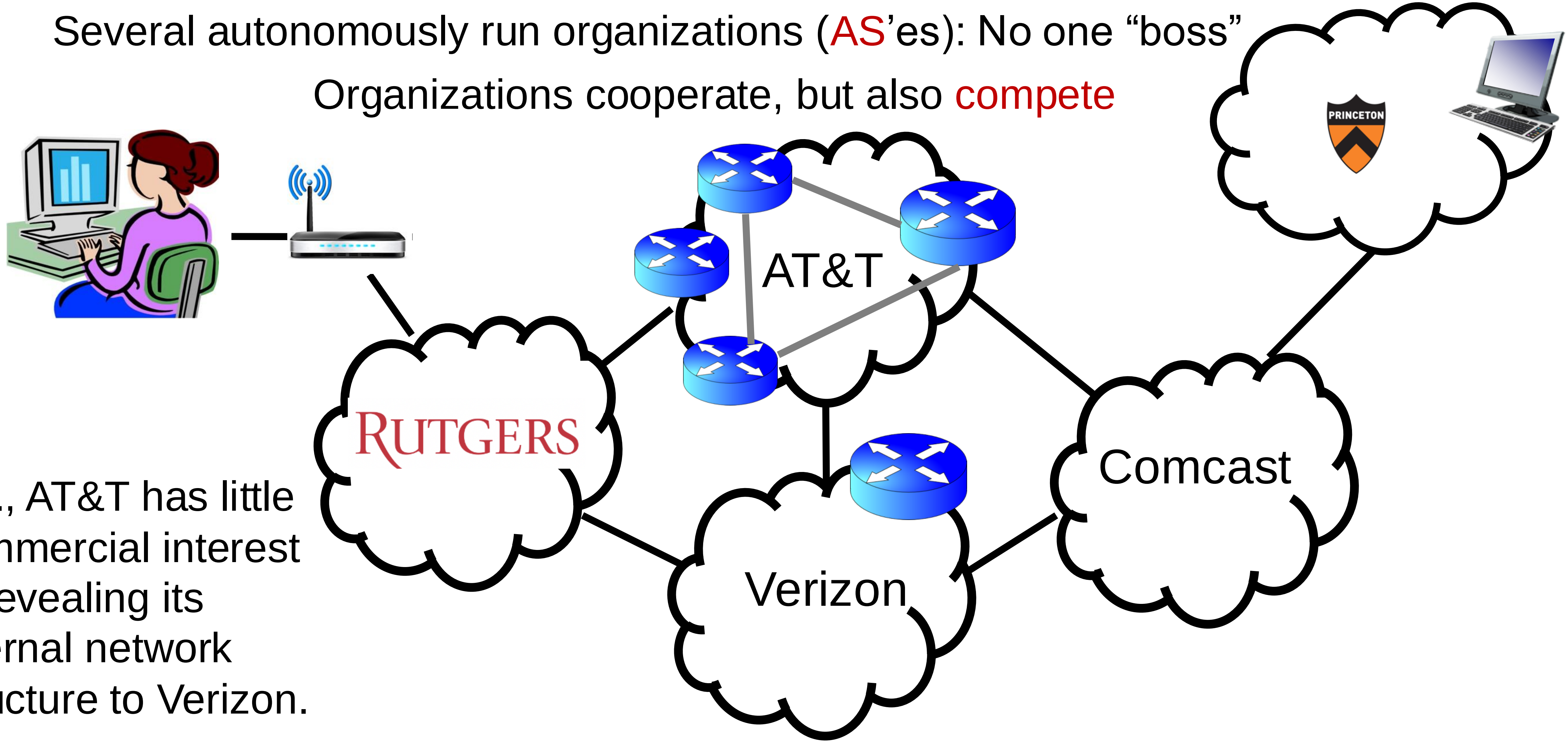
The Internet is a large federated network



The Internet is a large federated network

Several autonomously run organizations (AS'es): No one "boss"

Organizations cooperate, but also compete

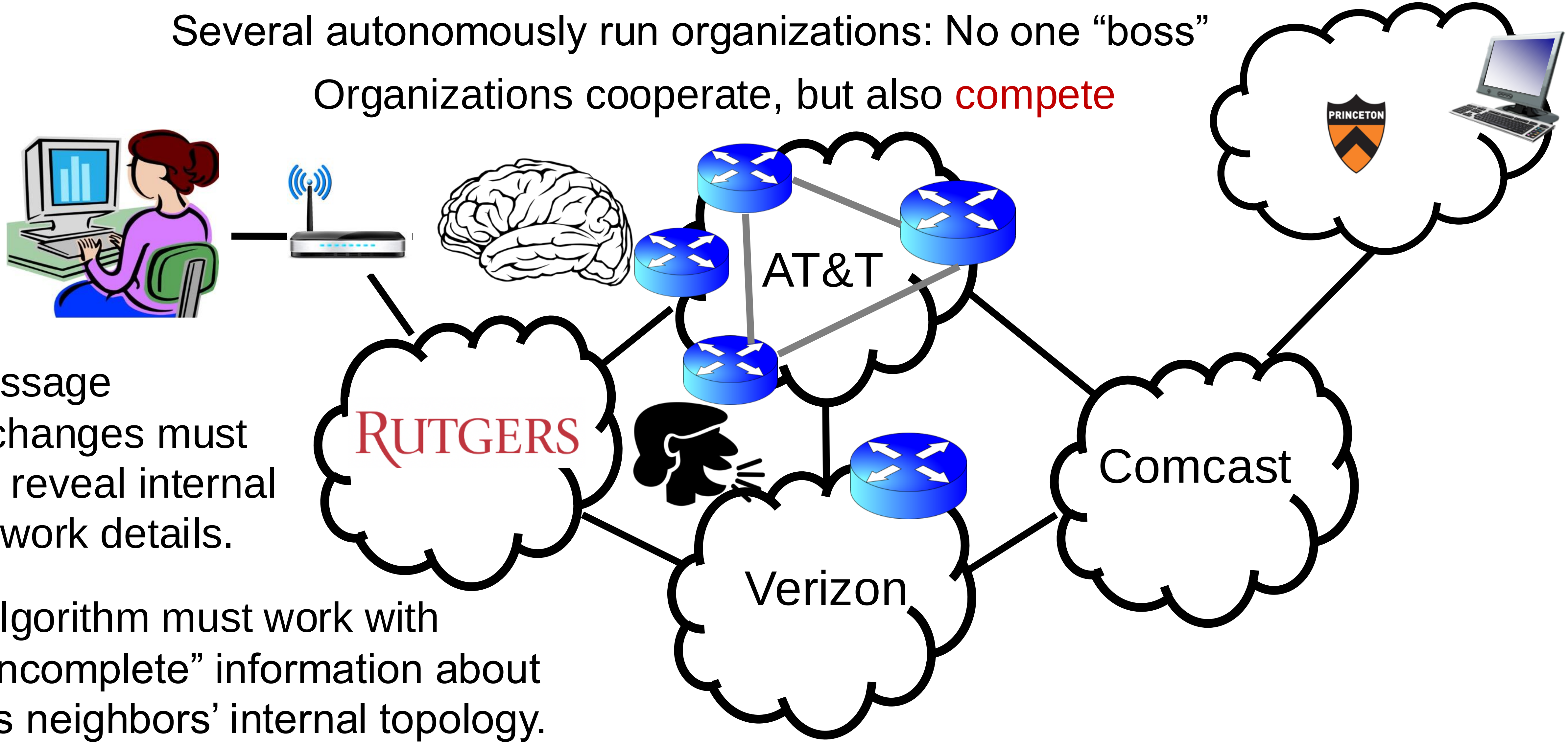


e.g., AT&T has little commercial interest in revealing its internal network structure to Verizon.

The Internet is a large federated network

Several autonomously run organizations: No one “boss”

Organizations cooperate, but also compete



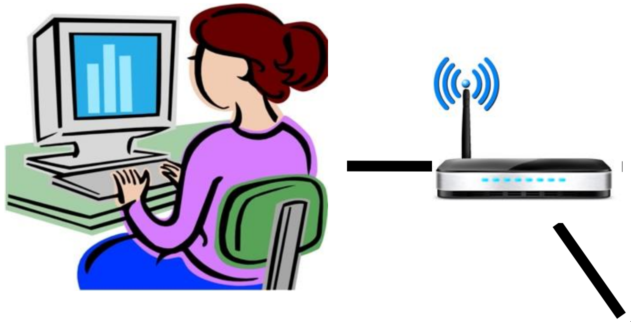
Message exchanges must not reveal internal network details.

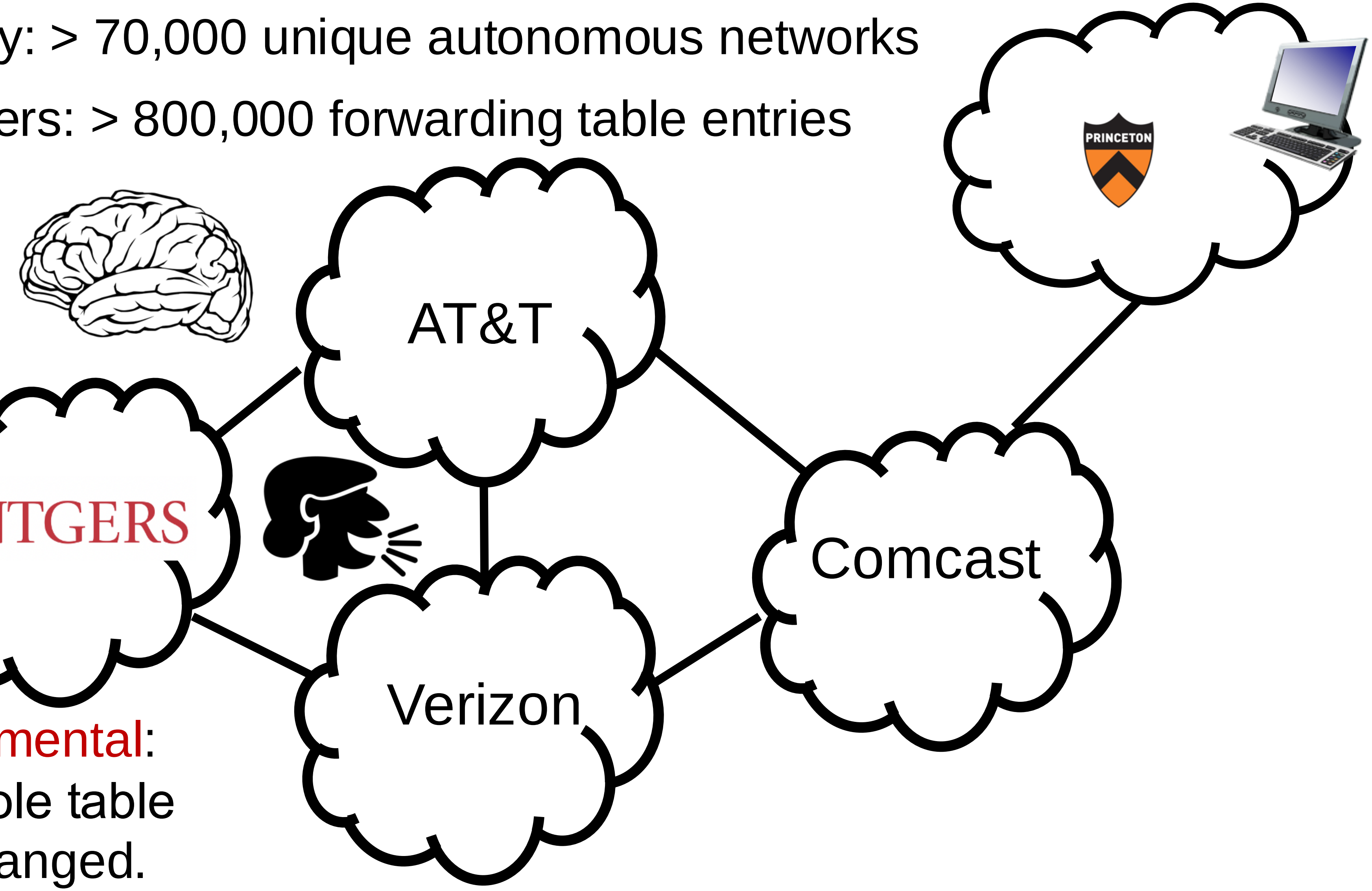
Algorithm must work with “incomplete” information about its neighbors’ internal topology.

The Internet is a large federated network

Internet today: > 70,000 unique autonomous networks

Internet routers: > 800,000 forwarding table entries



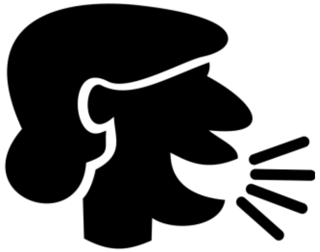


Keep messages & tables as small as possible. Don't flood

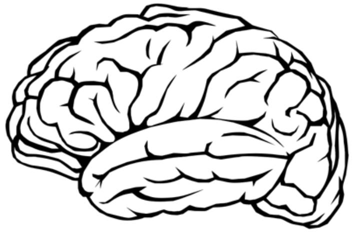
Algorithm must be incremental: don't recompute the whole table on every message exchanged.

Inter-domain Routing

- The Internet uses Border Gateway Protocol (BGP)
- All AS’es speak BGP. It is the glue that holds the Internet together
- BGP is a path vector protocol



Messages?



Algorithm?

Routing protocols

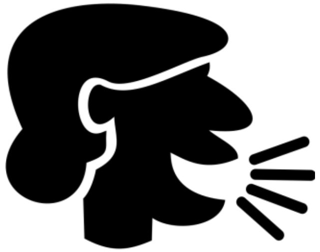
Link state protocols

Distance vector protocols

Path vector protocols

Applicable within a single AS, focus on efficiency

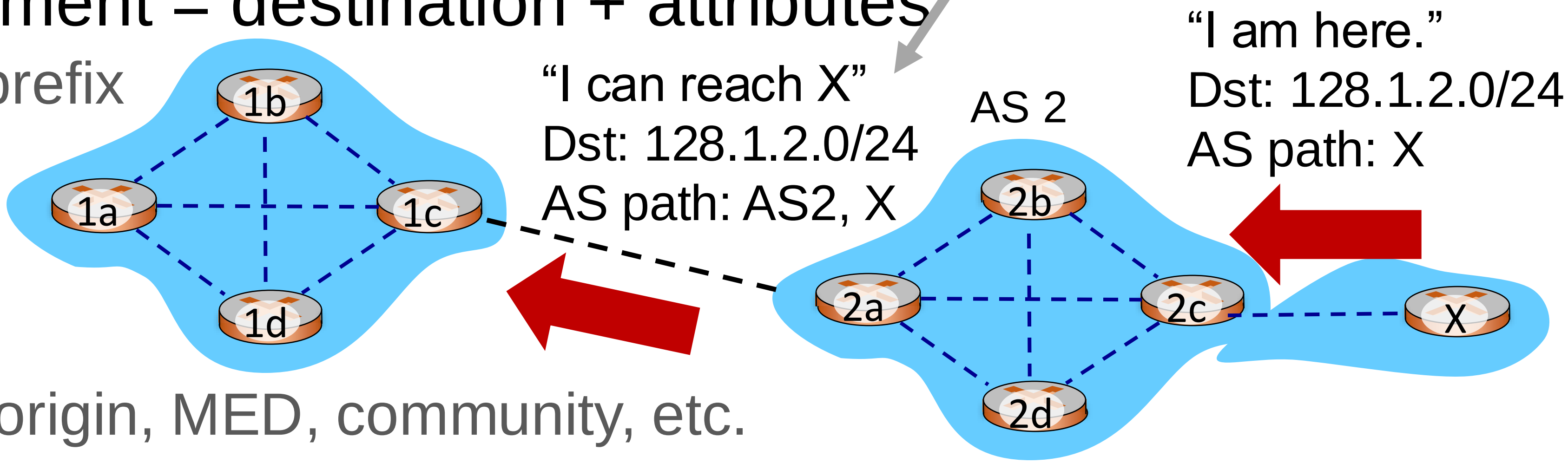
Q1. BGP Messages



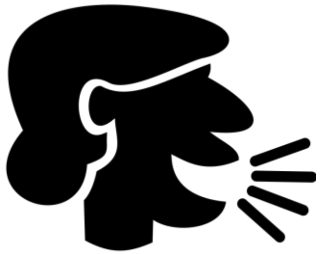
Exchange paths: path vector

No internal link or topology

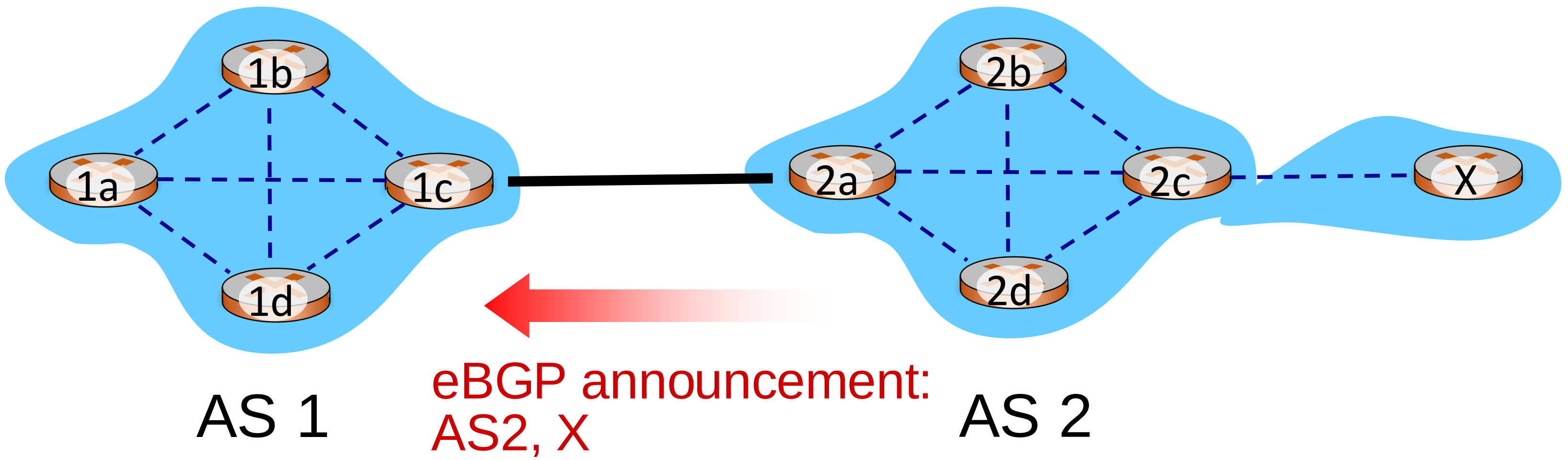
- Routing Announcements or Advertisements
  - “I am here” or “I can reach here”
  - Occur over a TCP connection (BGP session) between routers
- Route announcement = destination + attributes
  - Destination: IP prefix
- Route Attributes:
  - AS-level path
  - Next hop
  - Several others: origin, MED, community, etc.
- An AS promises to use advertised path to reach destination
- Only route changes are advertised after BGP session established



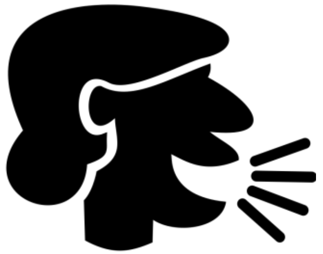
Q1. Next Hop



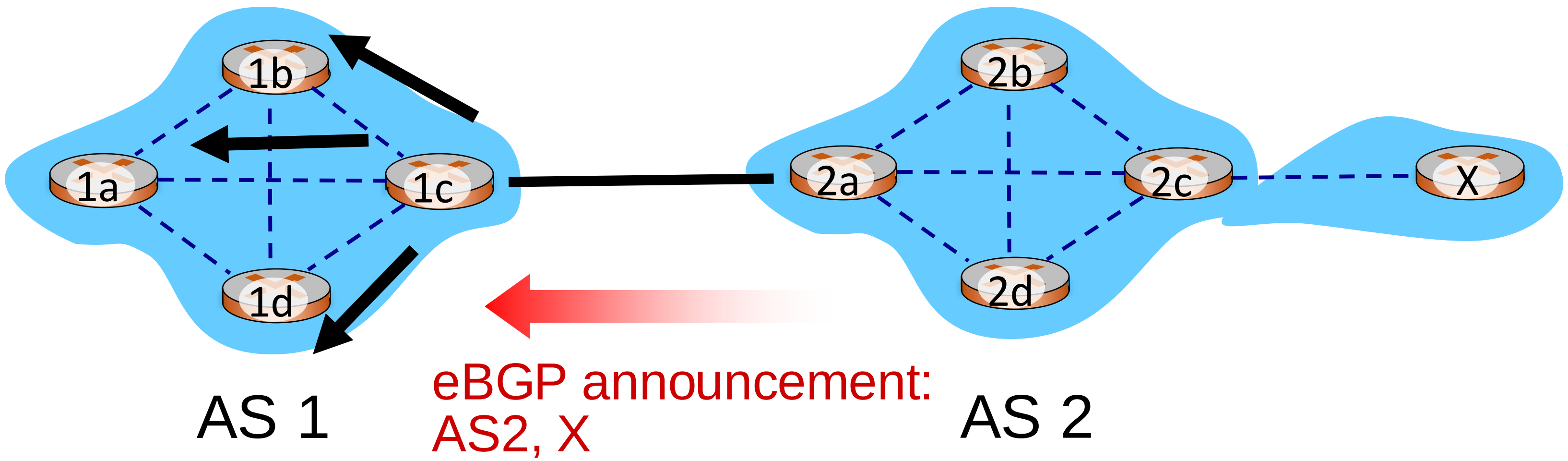
- Next hop conceptually denotes the first router interface that begins the AS-level path
  - The meaning of this attribute is context-dependent
- In an announcement arriving from a different AS (eBGP), next hop is the router in the next AS which sent the announcement
  - Example: Next Hop of the eBGP announcement reaching 1c is 2a



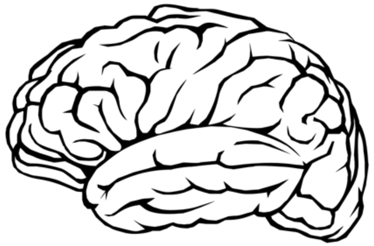
Q1. Next Hop



- Suppose router 1c receives a path advertisement
- Router 1c will propagate the announcement inside the AS using iBGP
- The next hop of this (iBGP) announcement is set to 1c
  - In particular, the next hop is an AS1 internal address



Q2. The algorithm

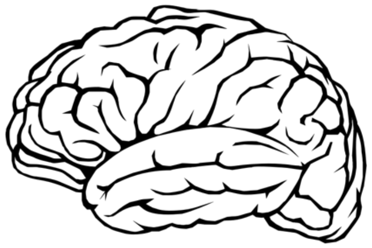


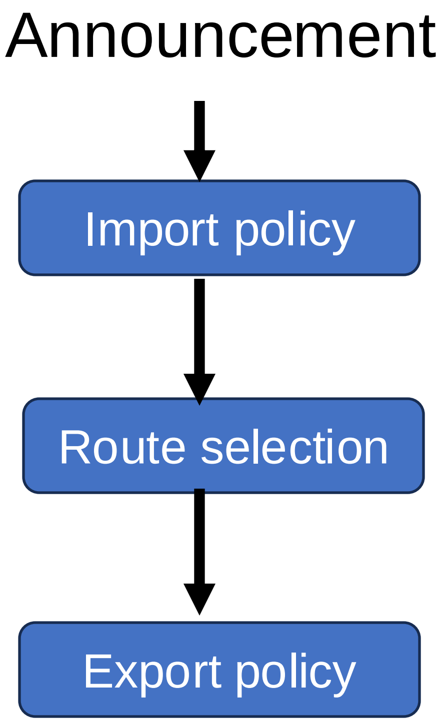
- A BGP router does *not* consider every routing advertisement it receives by default to make routing decisions
  - An import policy determines whether a route is even considered a candidate
- Once imported, the router performs route selection
- A BGP router does *not* propagate its chosen path to a destination to all other AS'es by default
  - An export policy determines whether a (chosen) path can be advertised to other AS'es and routers

Programmed by network operator

Policy considerations make BGP paths very different from "the most efficient" paths

Policies in BGP

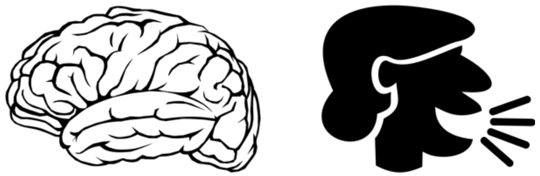


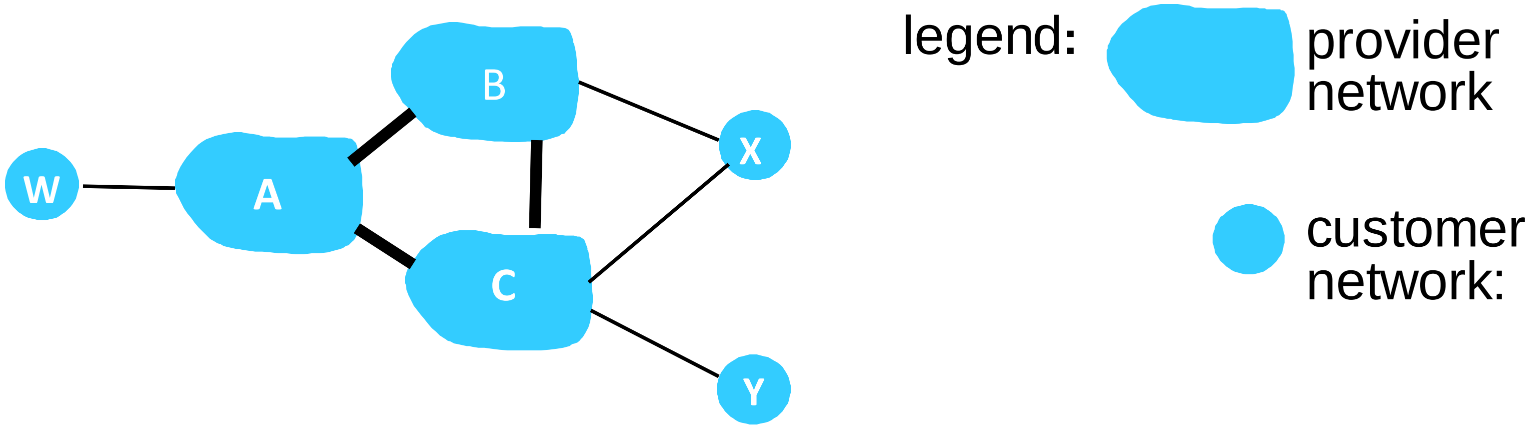


Policy arises from business relationships

- Customer-provider relationships:
  - E.g., Rutgers is a customer of AT&T
- Peer-peer relationships:
  - E.g., Verizon is a peer of AT&T
- Business relationships depend on where connectivity occurs
  - “Where”, also called a “point of presence” (PoP)
  - e.g., customers at one PoP but peers at another
  - Internet-eXchange Points (IXPs) are large PoPs where ISPs come together to connect with each other (often for free)

BGP Export Policy

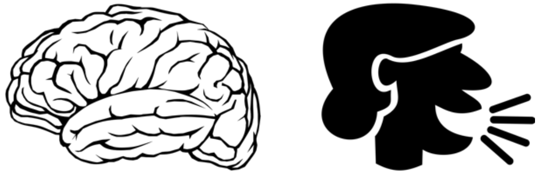


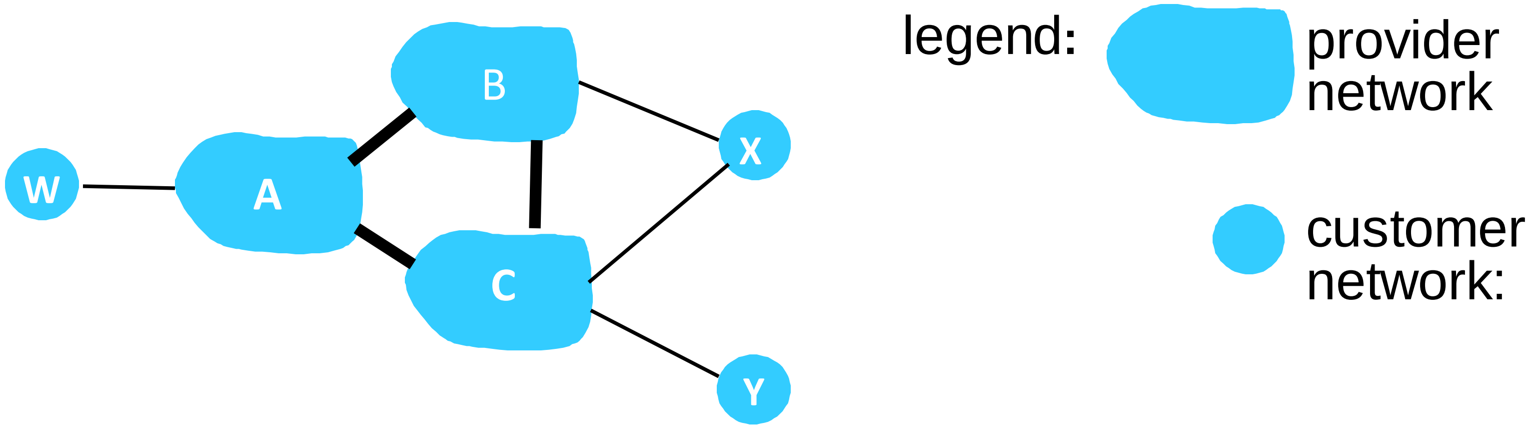


Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A,B,C are provider networks
- X,W,Y are customers (of provider networks)
- X is dual-homed: attached to two networks
- policy to enforce: X does not want to route from B to C via X
  - So, X will not announce to B a route to C

BGP Export Policy





Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A announces path Aw to B and to C
- B will not announce BAw to C:
  - B gets no "revenue" for routing CBAw, since none of C, A, w are B's customers
- C will route CAw (not using B) to get to w

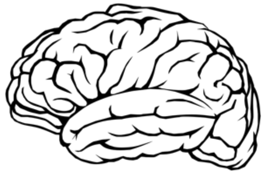
Impact of BGP export policies

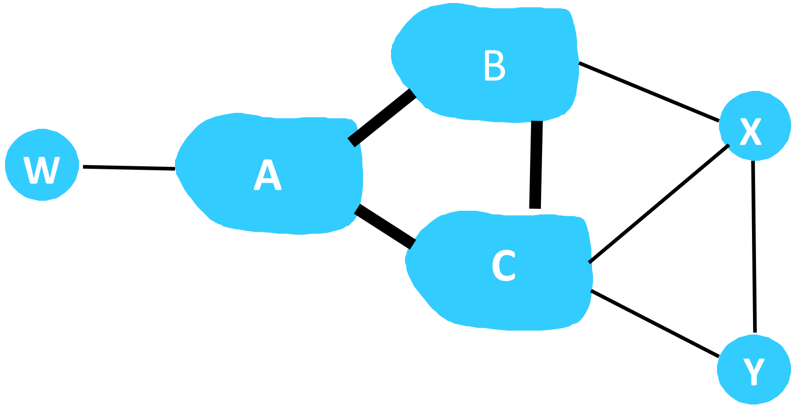
- Based on your location on the Internet,
- Some paths aren’t visible or usable, even if they physically exist
- Many efficient paths could be eliminated from actual use
- Focus on financial incentives, not efficient end-to-end paths

BGP Import Policy

- Remove common misconfigurations or problematic routes
- Loops
- Too-specific prefixes (e.g., anything longer than /24)

BGP Import Policy





legend:

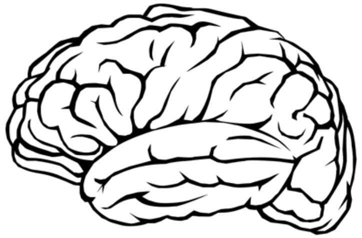
provider network

customer network:

Suppose an ISP wants to minimize costs by avoiding routing through its providers when possible.

- Suppose C announces path Cy to x
- Further, y announces a direct path (“y”) to x
- Then x may choose not to import the path Cy to y since it has a peer path (“y”) towards y

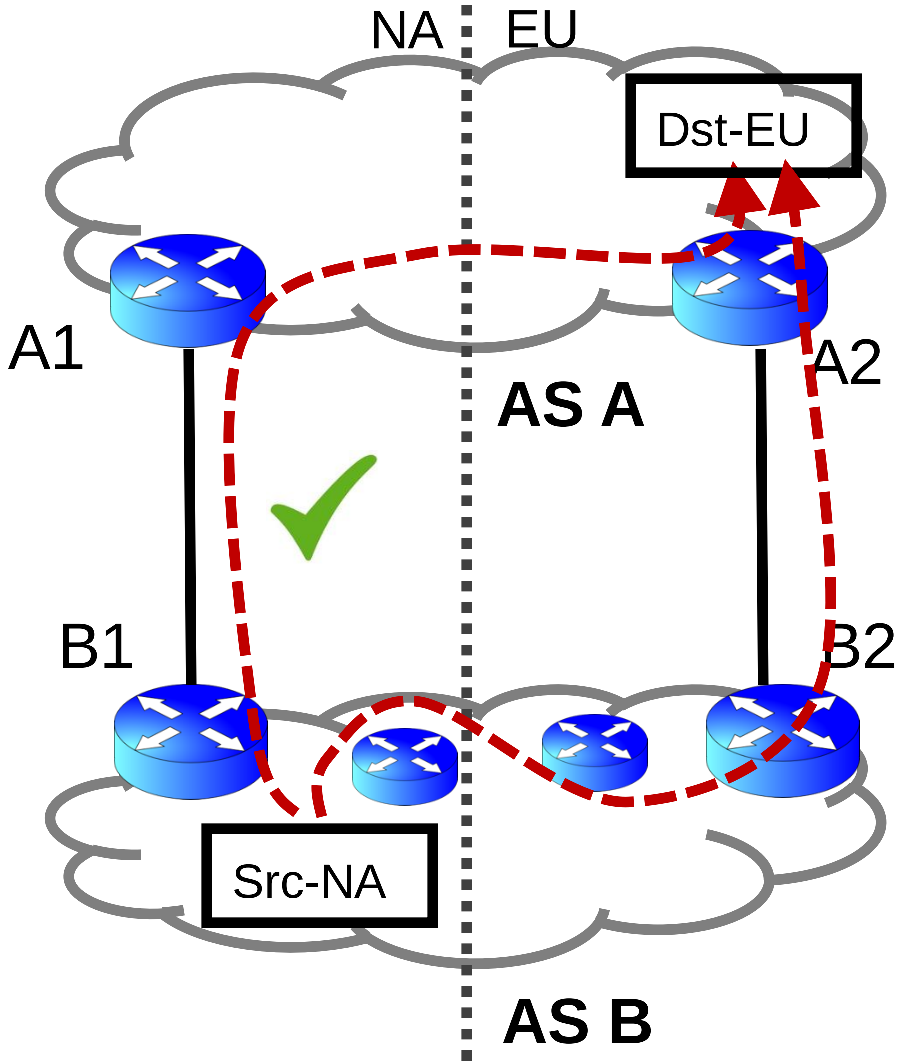
Q2. BGP Route Selection



- When a router imports more than one route to a destination IP prefix, it selects route based on:
  1. local preference value attribute (import policy decision -- set by network admin)
  2. shortest AS-PATH
  3. closest NEXT-HOP router
  4. Several additional criteria: You can read up on the full, complex, list of criteria, e.g., at https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

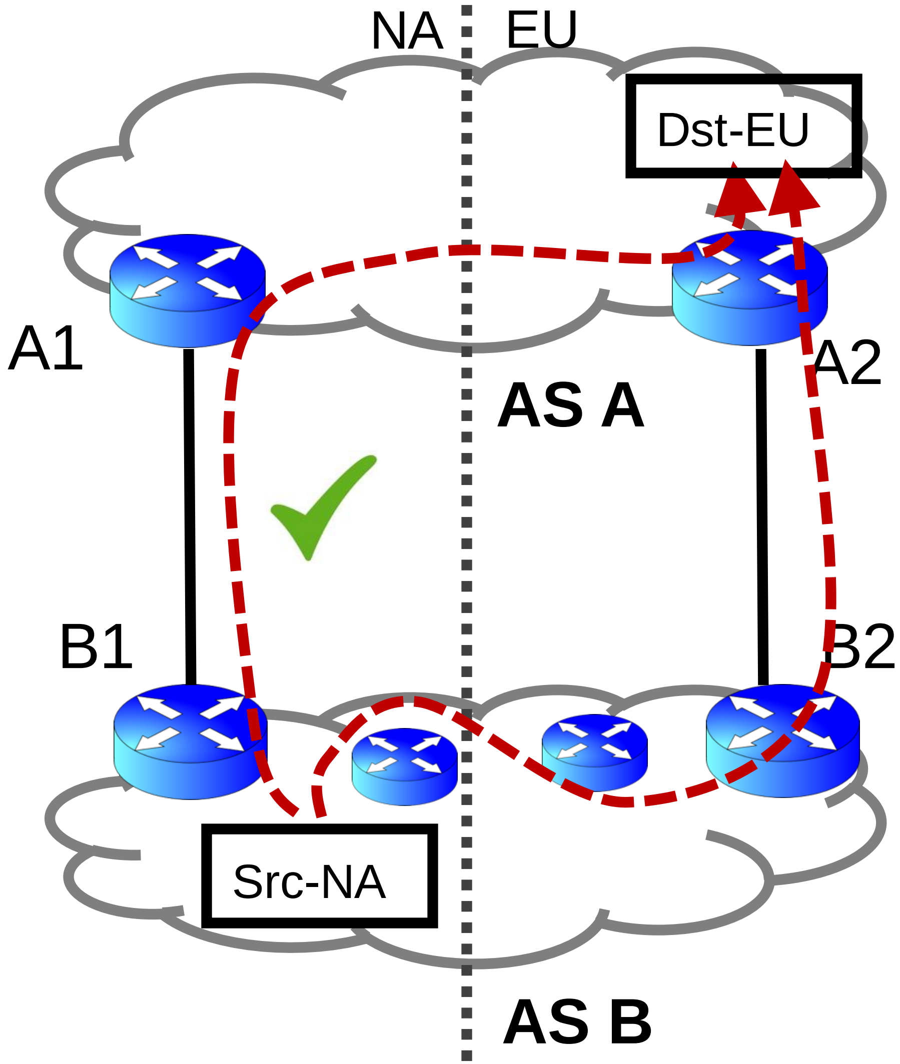
Example of route selection

- Suppose AS A and B are connected to each other both in North America (NA) and in Europe (EU)
- A source in NA wants to reach a destination in EU
- There are two paths available
  - *Assume* same local preference
  - Same AS path length
- Closest next hop-router: choose path via B1 rather than B2



Example of route selection

- Choosing closest next-hop results in early exit routing
  - Try to exit the local AS as early as possible
  - Also called hot potato routing
- Reduce resource use within local AS
  - potentially at the expense of another AS
- A potential reason for inefficiency



BGP Path Selection

Approaches to bring flexibility:
Flexible control logic for path selection (Google, Facebook)
Detour/overlay routing (Akamai)

- Local preference, shortest AS path, closest NEXT HOP, etc.
- Not capacity aware
- Not performance aware
- Not aware of the length of the path (in # routers)
  - The protocol does not even incorporate precise perf/capacity info
- Financial incentive, not end-to-end performance, heavily determines peering and capacity
- Only a single path per destination
- Can be slow to converge
- Vulnerable to bugs and malice

Application Architecture

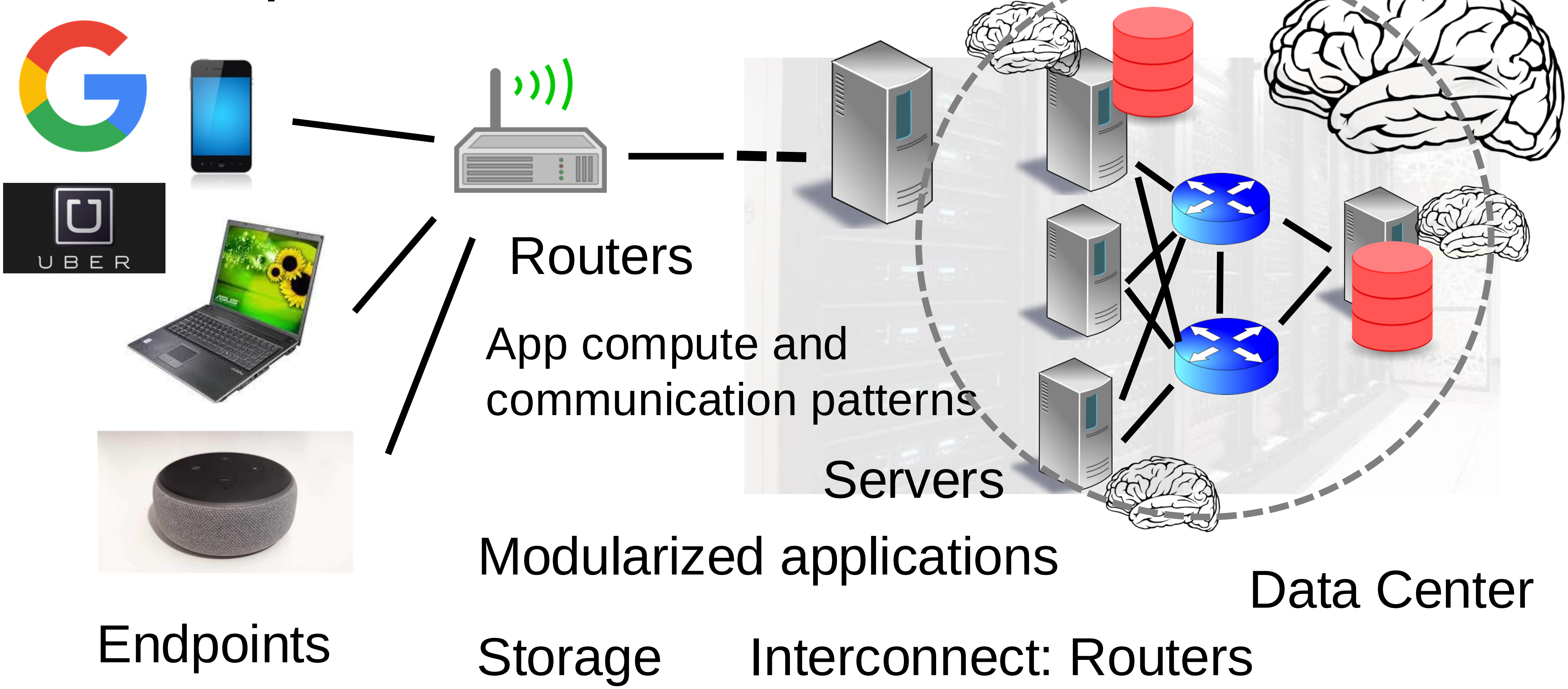
Lecture 4b

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S25

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Components of an Internet Service



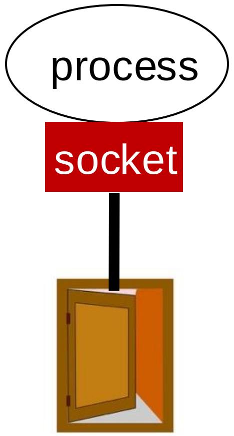
Web Servers

Often the first app point where a user request lands

Parse HTTP request

```
GET / HTTP/1.1
Host: example.com
```

(many other headers!)



IP_B + $port_B$

`bind(IPaddr`$_B$`, port`$_B$`)`

`listen()`

`accept()`

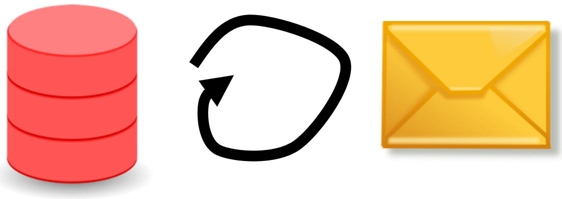
`recv()/send()/..`

Find a file, run a script, …

```
HTTP/1.1 200 OK
Content-Type: text/html
```

Send response header

Read file, `send()` data



Overloaded with functionality

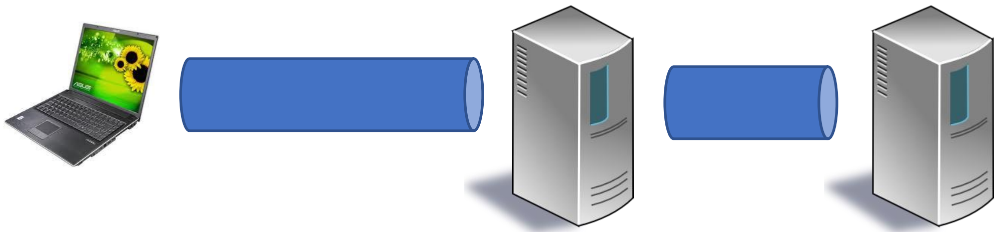
Often the first app point where a user request lands

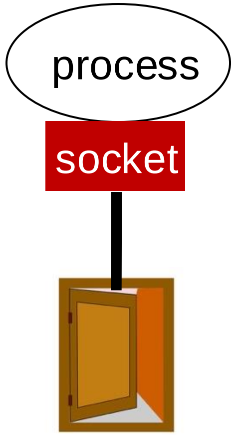
Find a file, run a script, …

$\underbrace{\hspace{10em}}$

Scripting: Python/PHP/nodejs fastCGI

Reverse proxy





IP_B + $port_B$

```
bind(IPaddr_B, port_B)
listen()
accept()
recv()/send()/..
```

Caching

Compression

Access control

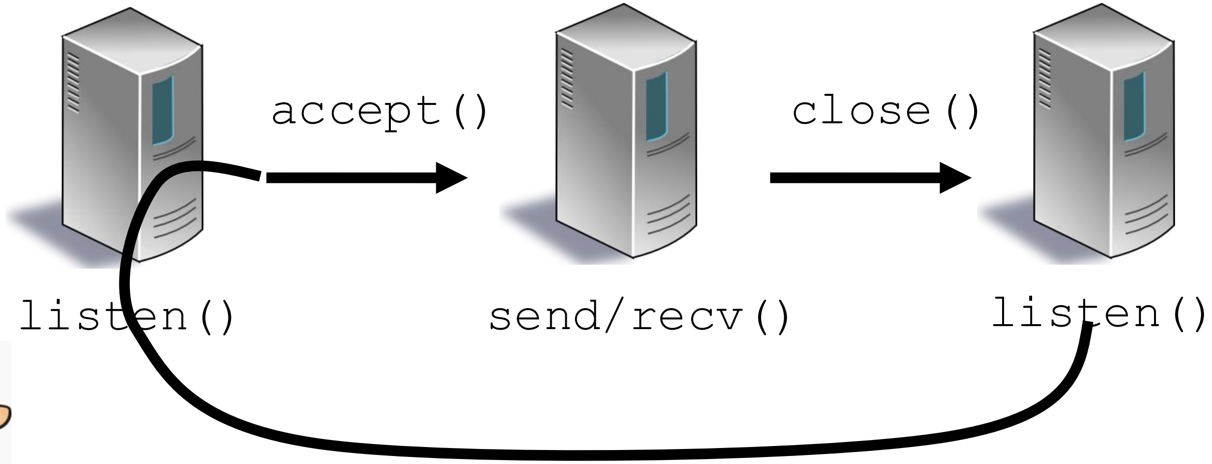
TLS

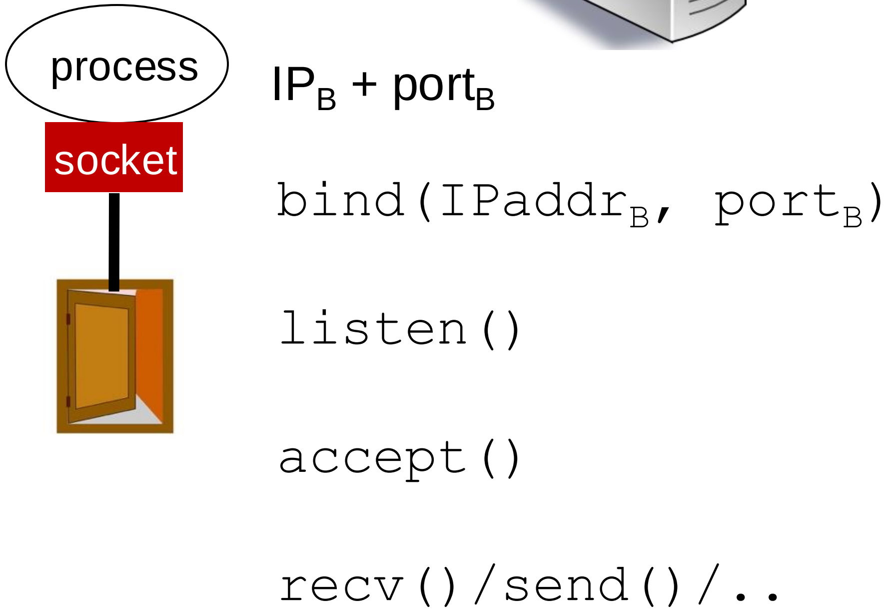
Media streaming

Image transforms

How does one design a web server?

- Process connections one at a time?





Many other requests waiting in the meanwhile

Powerful server doing nothing most of the time

How does one design a web server?

- Process other requests while waiting for one to finish

process

socket

IP_B + $port_B$

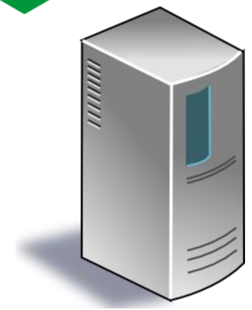
`bind(IPaddr_B, port_B)`

`listen()`

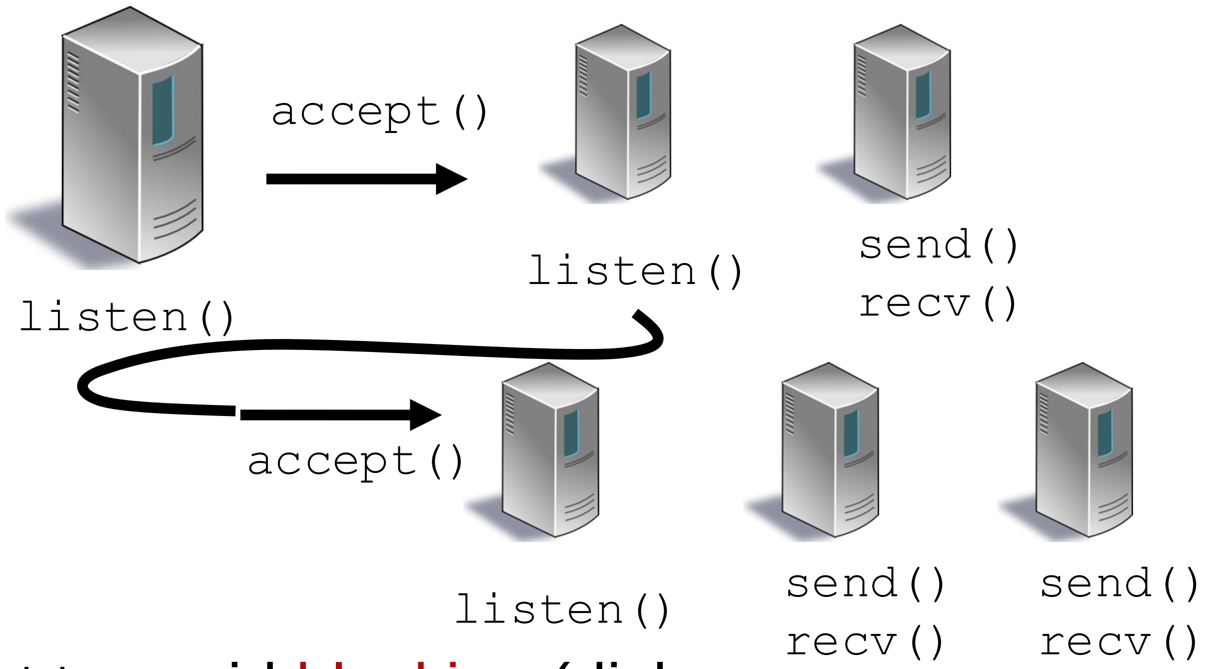
`accept()`

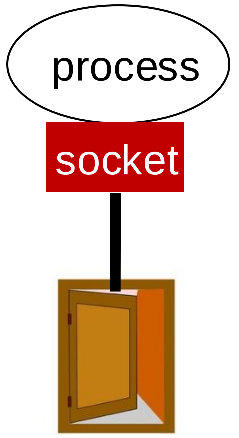
`recv()/send()/..`

Parallelism



- Process requests in parallel with other requests
- One design: multiprocessing/multithreading (MP/MT)





IP_B + $port_B$

```
bind(IPaddr_B, port_B)

listen()

accept()
fork()
recv()/send()/..
```

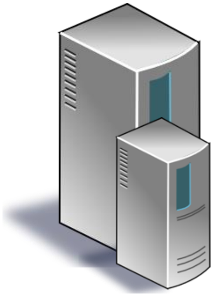
Great to avoid blocking (disk I/O, fastCGI, …)

Overhead grows with # connections: more / longer lived

Concurrency

State of the art designs combine parallelism (multiprocess/thread) with concurrency (event-driven)

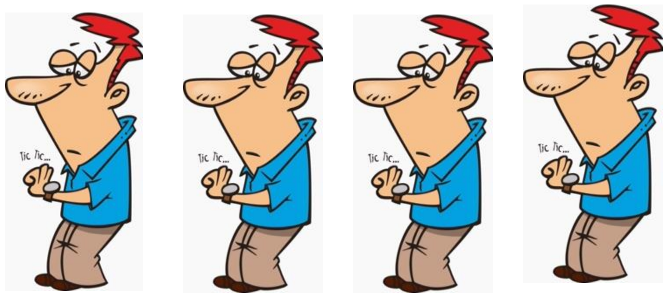
- Process other requests while waiting for one to finish
- A different design: single process event driven (SPED)



`accept()`

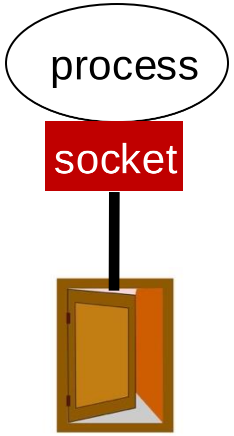
`recv()`

A queue of events



`epoll, select, kqueue, io_uring`

Lightweight



IP_B + $port_B$

`bind(IPaddr_B, port_B)`

`listen()`

`accept()`

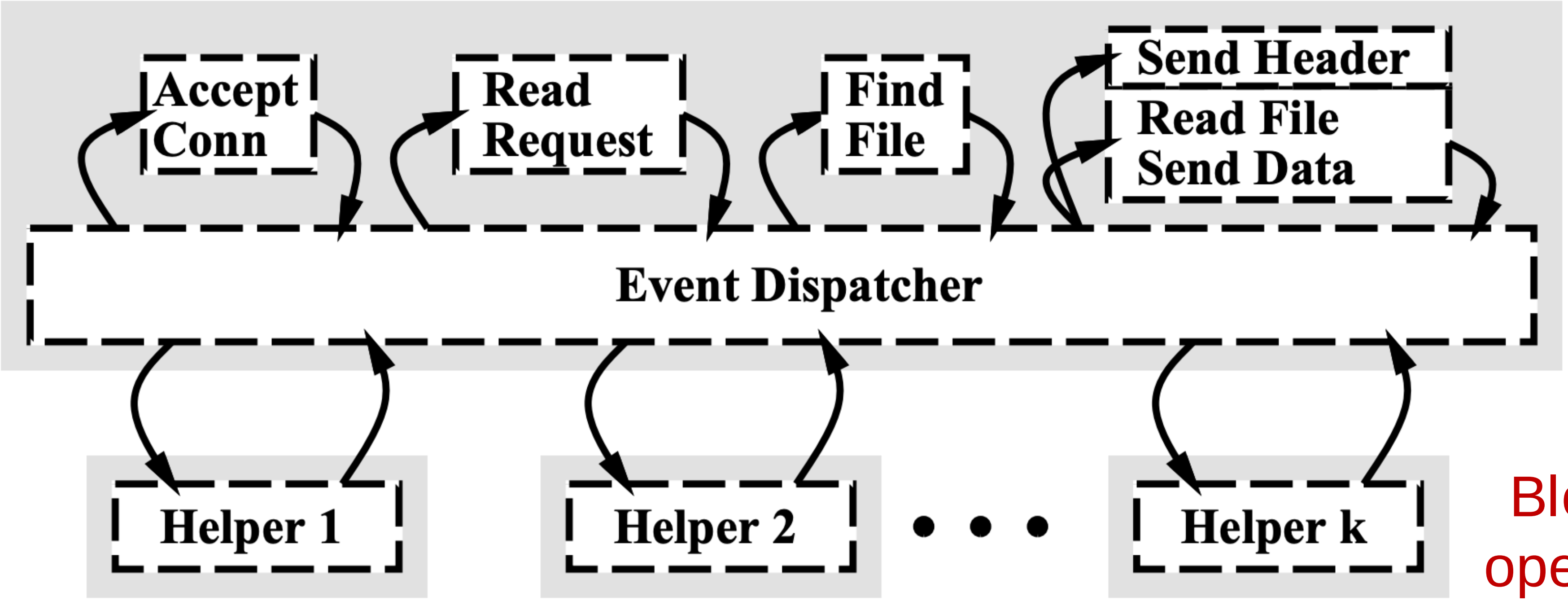
`recv()/send()/..`

Can block if any of the requests block (asynchronous IO support can be incomplete & complex)

Avoid overheads of multiple processes and threads https://aosabook.org/en/v2/nginx.html

Using parallelism + concurrency

- Asymmetric Multi-Process Event Driven (AMPED)

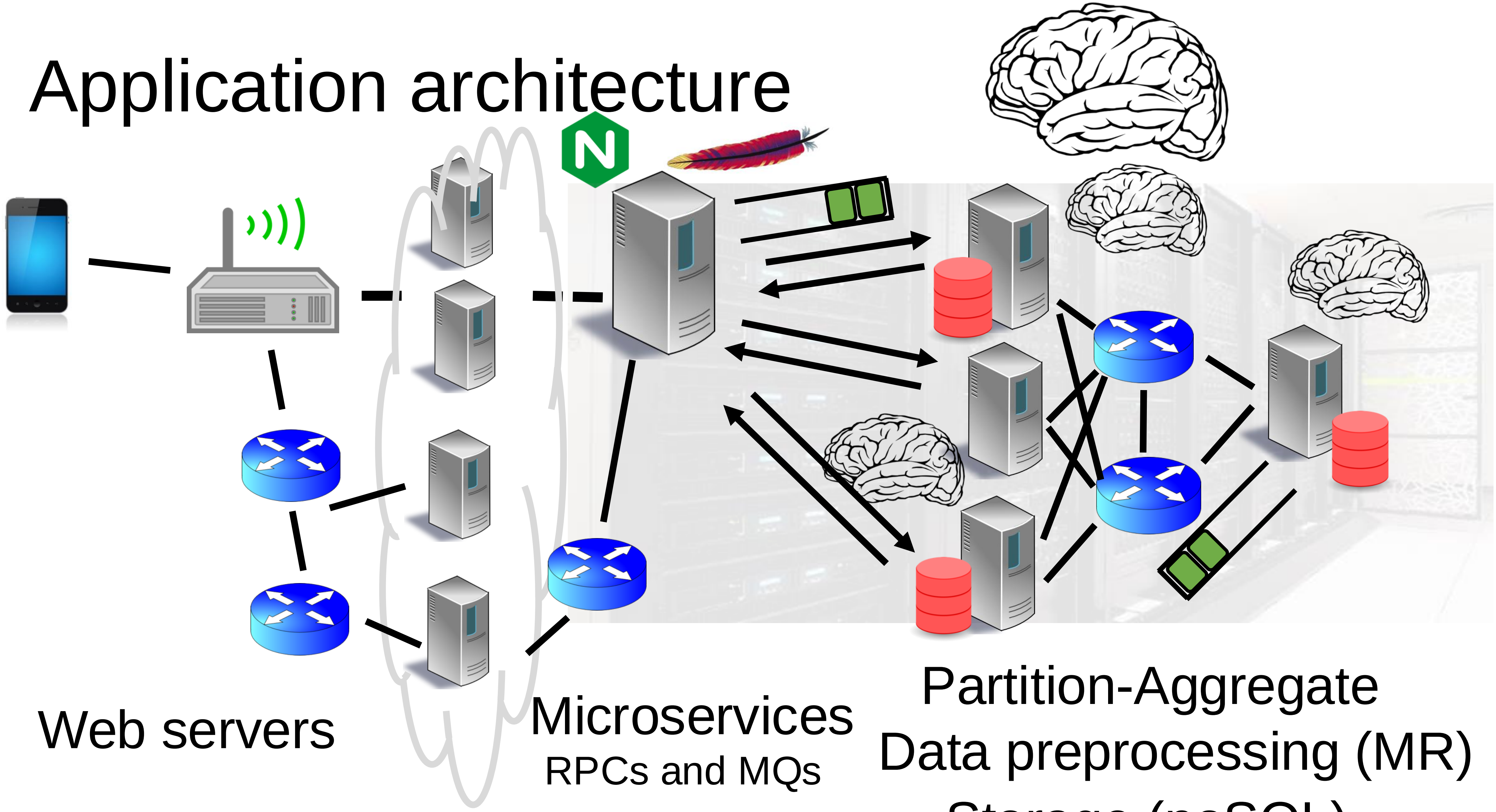


Event-driven main process

Blocking operations

Flash: An efficient and portable Web server

Application architecture



Web servers

Microservices

RPCs and MQs

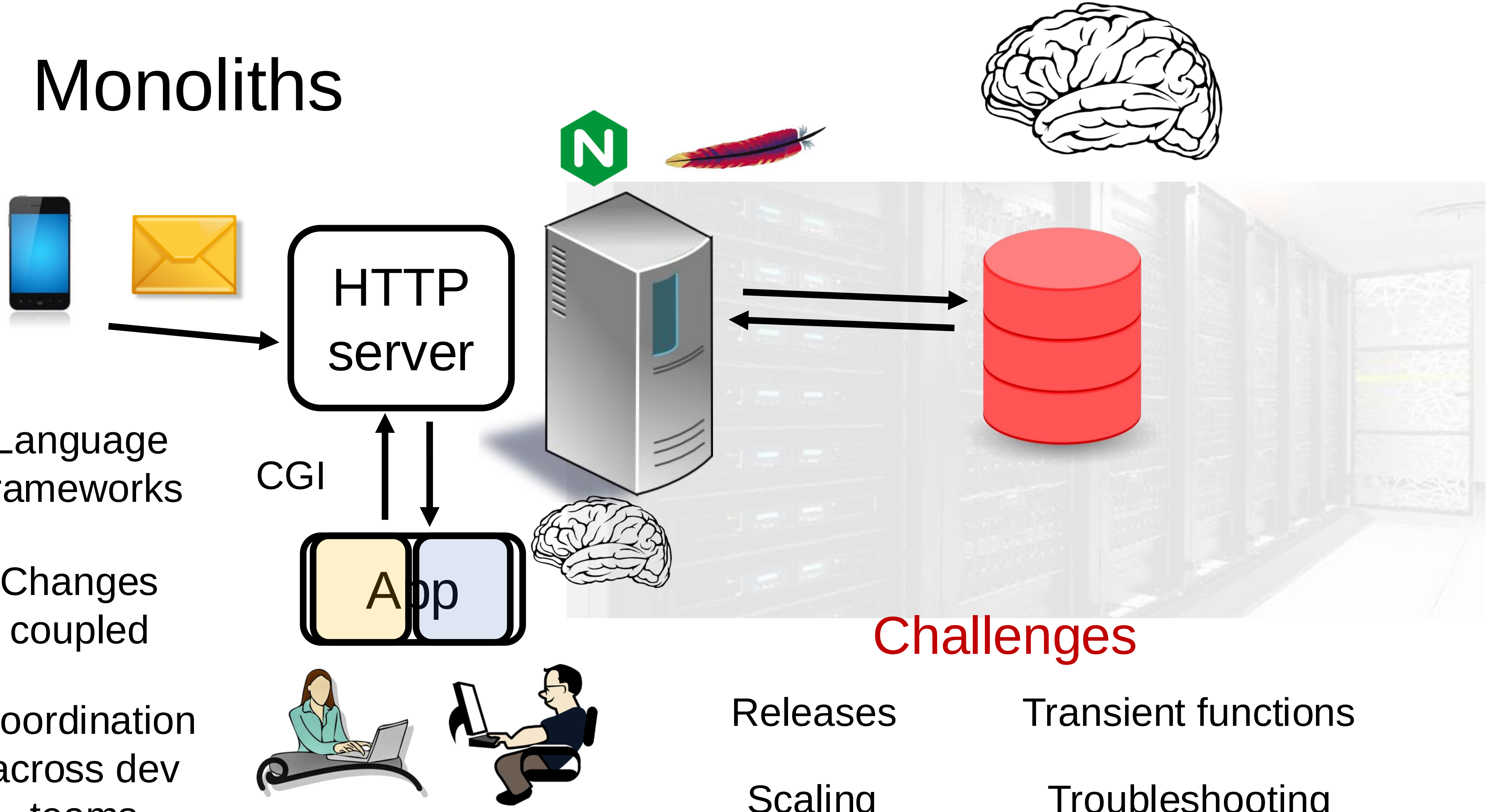
Partition-Aggregate

Data preprocessing (MR)

Storage (noSQL)

Microservice Architectural Pattern

Monoliths



Language frameworks

Changes coupled

Coordination across dev teams

Challenges

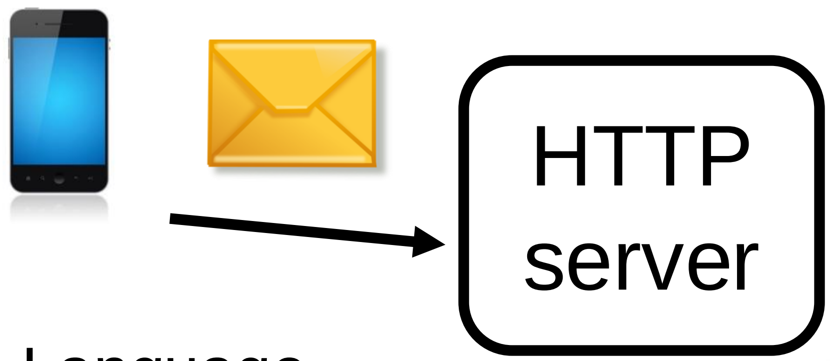
Releases

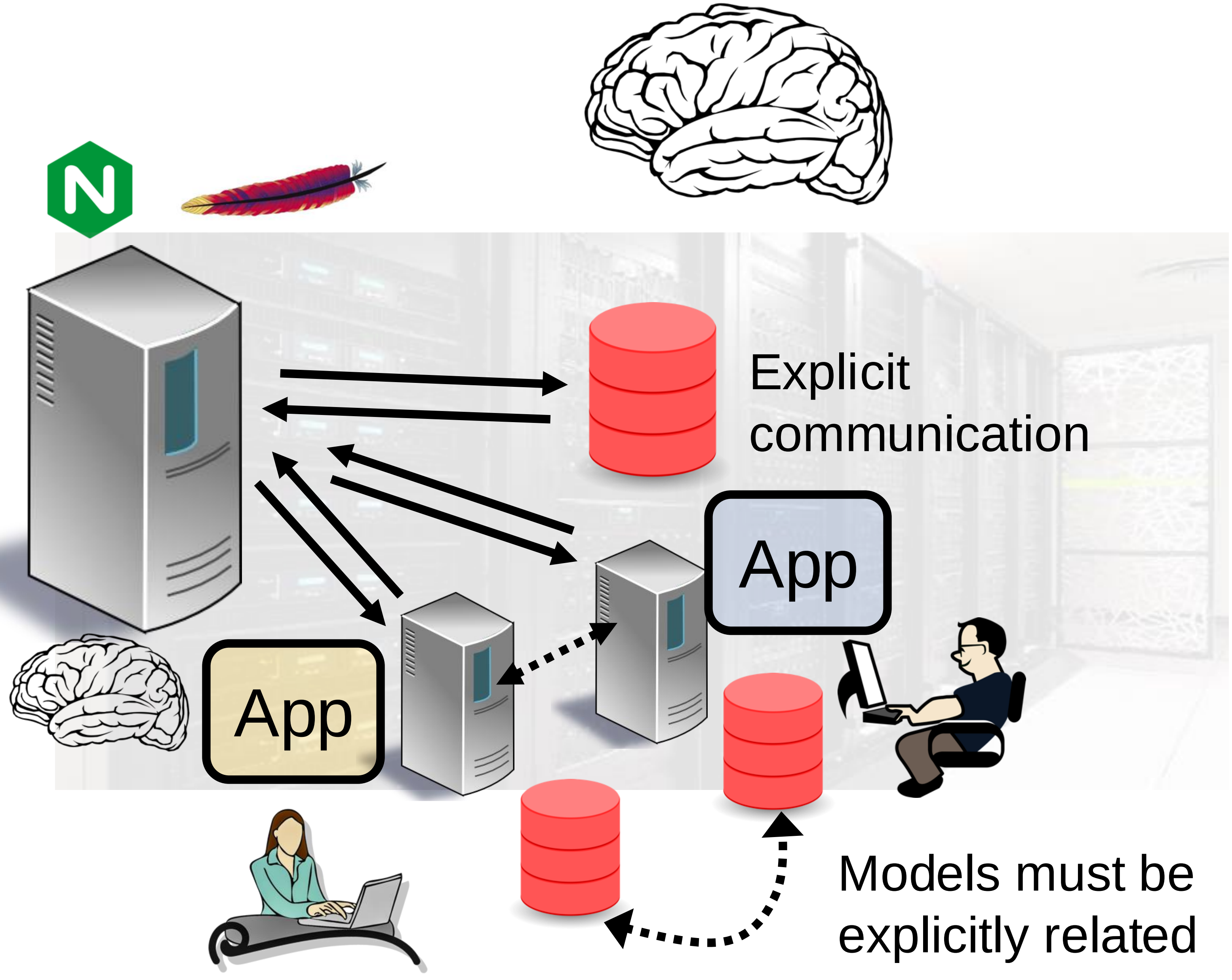
Transient functions

Scaling

Troubleshooting

Microservices





Language
of choice

Independently
upgrade and
deploy

Independent/de-
centralized data
models & storage

Distinct
from a
software
library:
Out of
process.

Loosely-coupled services

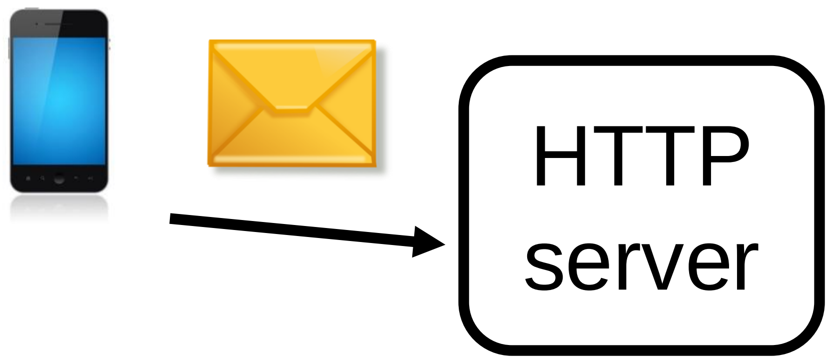
In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

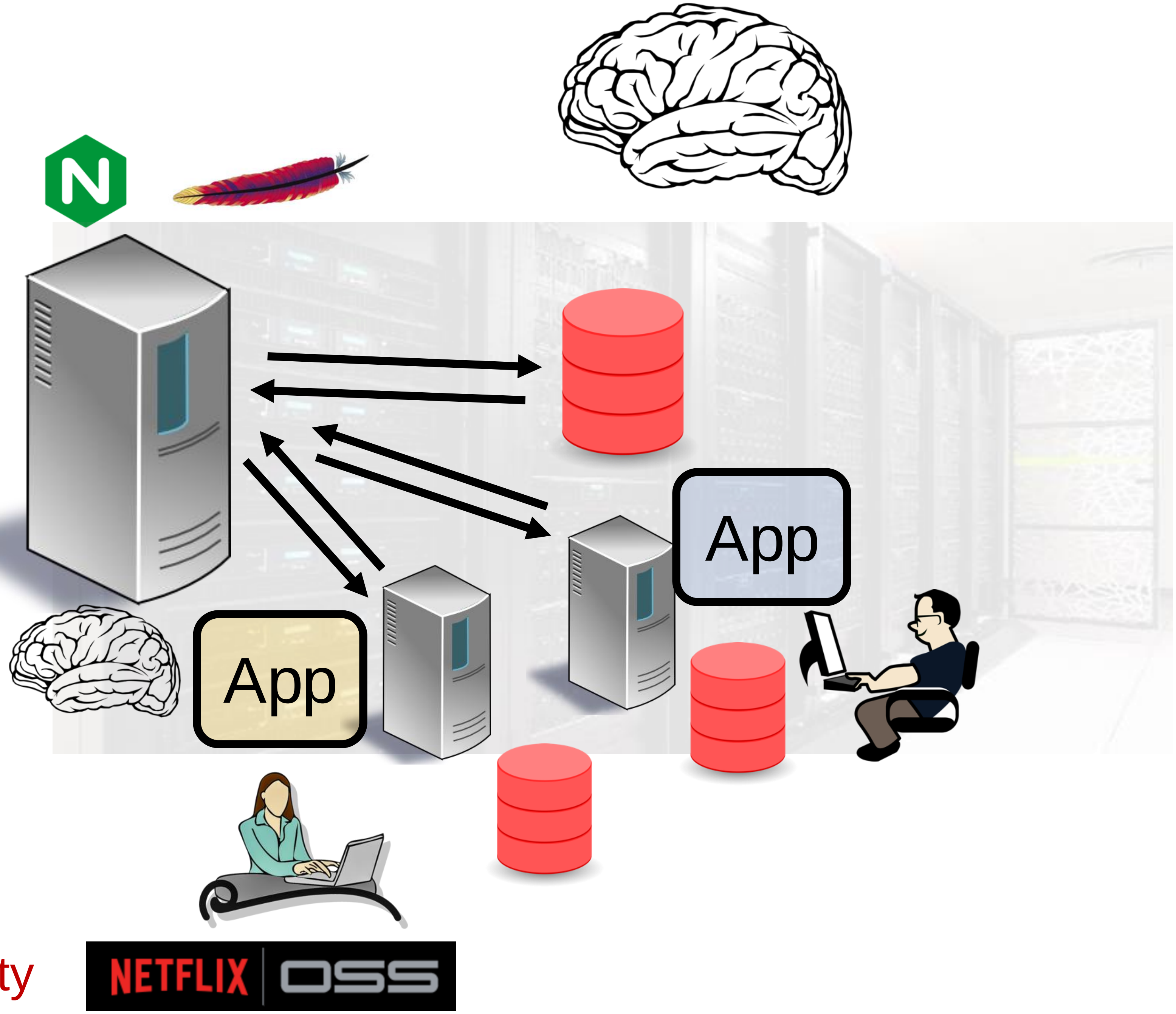
-- James Lewis and Martin Fowler (2014)

https://martinfowler.com/articles/microservices.html

https://martinfowler.com/microservices/

How to split?





Business Capabilities

Boundaries of change

Lifetime of the service?

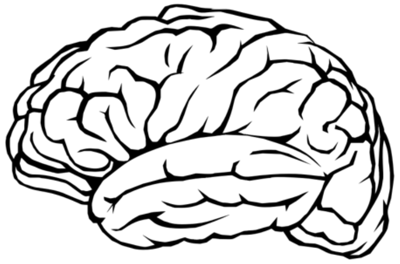
Who should know (or not)?

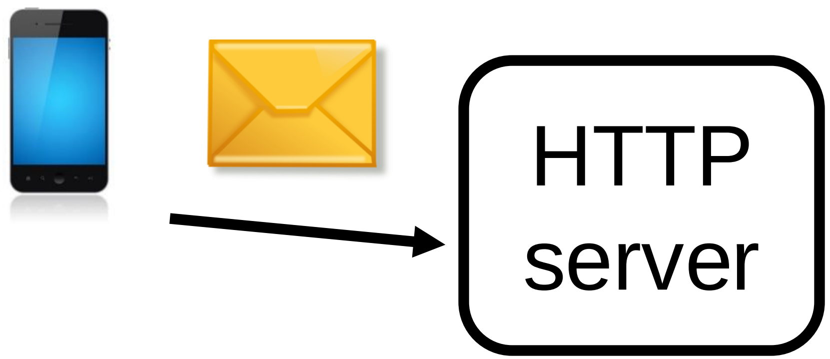
Changing together vs separately?

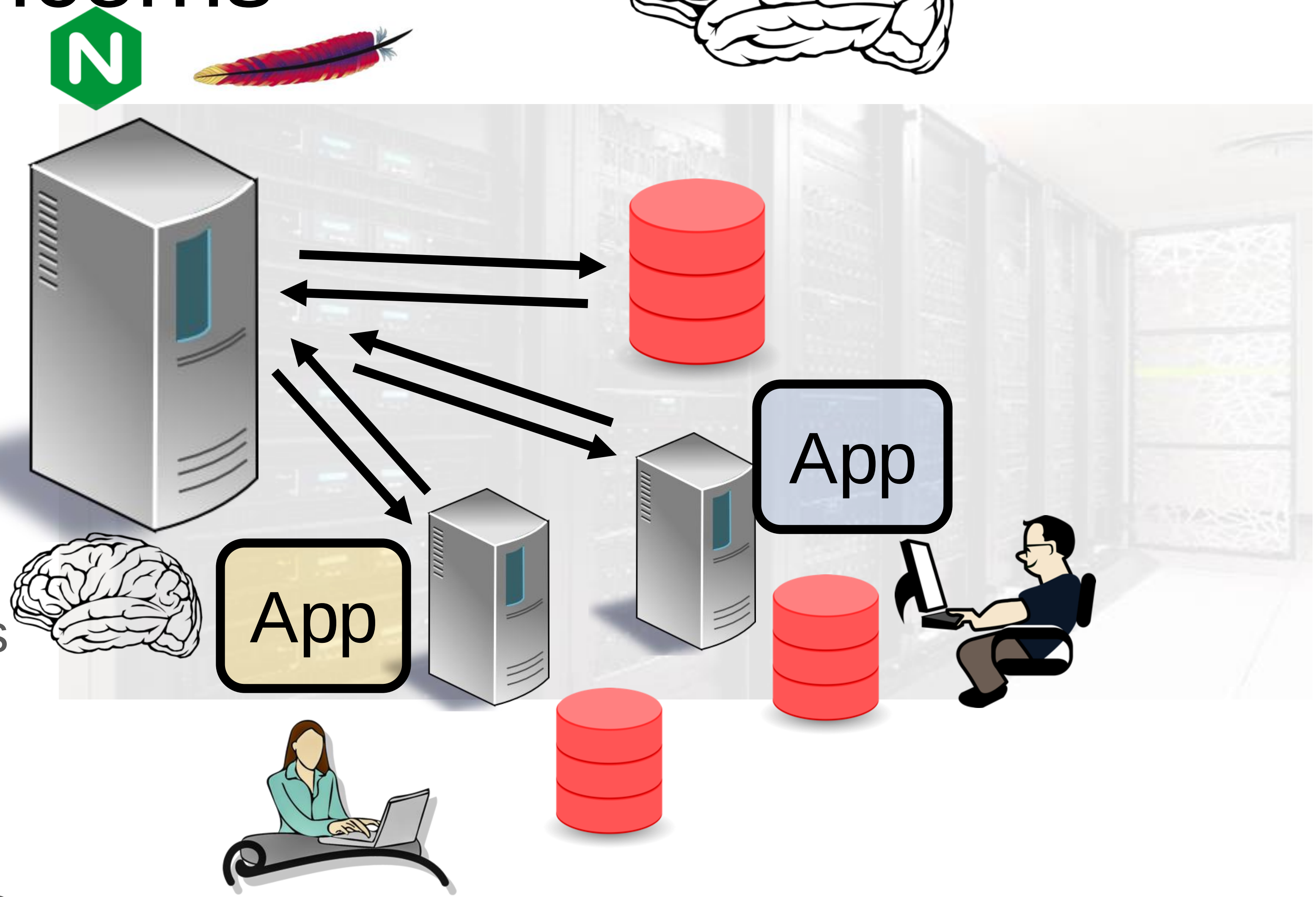
Heterogeneity in resource use

Refactoring common functionality

Salient new concerns







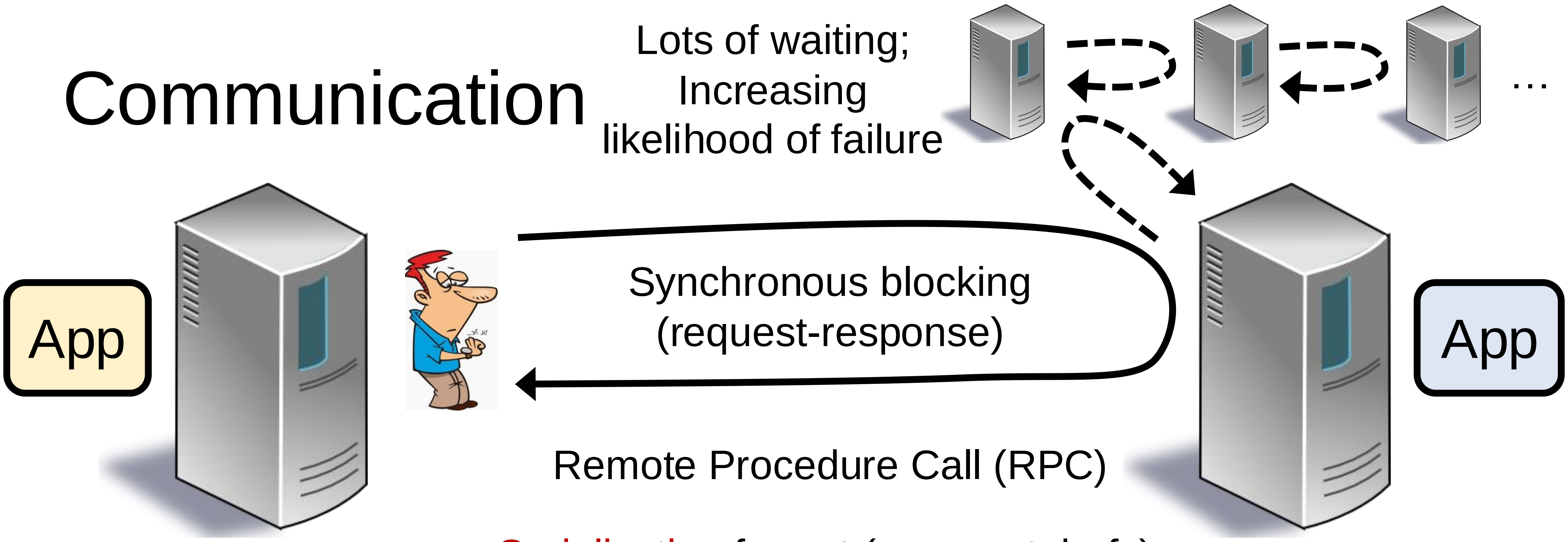
Communication

No longer a function call
Design good module boundaries

Failures

Networks & components fail
No longer in the same process

Communication

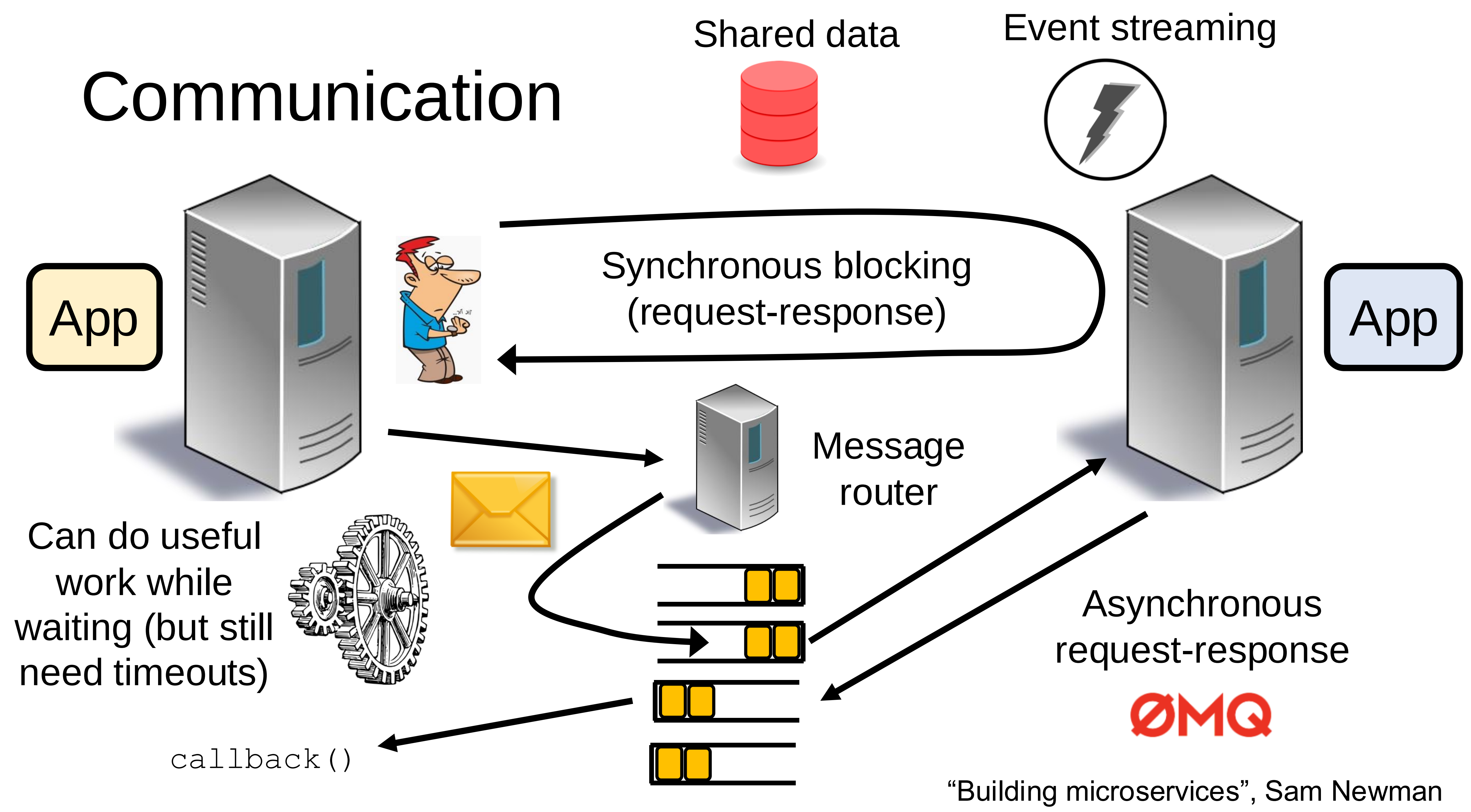


Serialization format (e.g., protobufs)

```
struct customer {
  string name;
  int customer_id;
  ..;
}
```

01101010101…
JSON
XML

```
struct customer {
  string name;
  int customer_id;
  ..;
}
```


Communication
Shared data
Event streaming
App
App
Synchronous blocking
(request-response)
Message
router
Can do useful
work while
waiting (but still
need timeouts)
callback()
Asynchronous
request-response
ØMQ
“Building microservices”, Sam Newman

Cost of communication: Performance

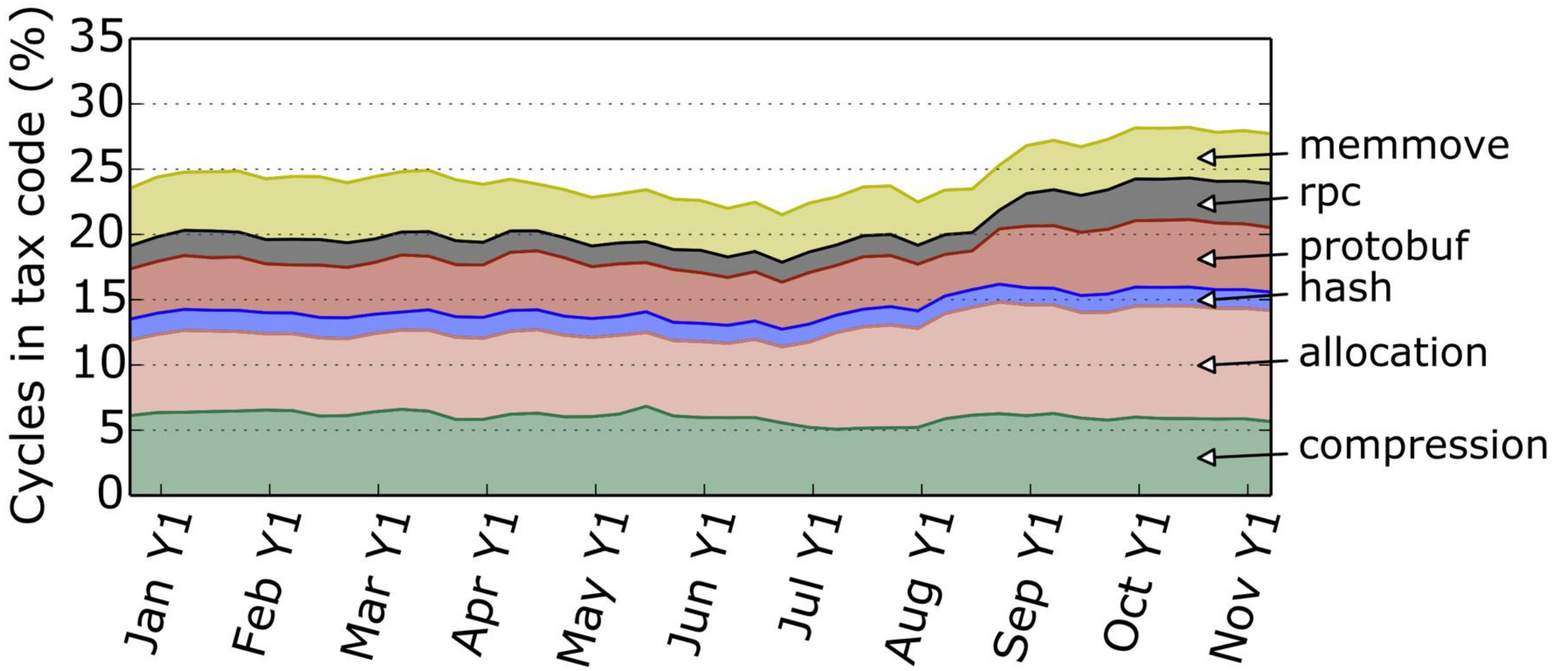
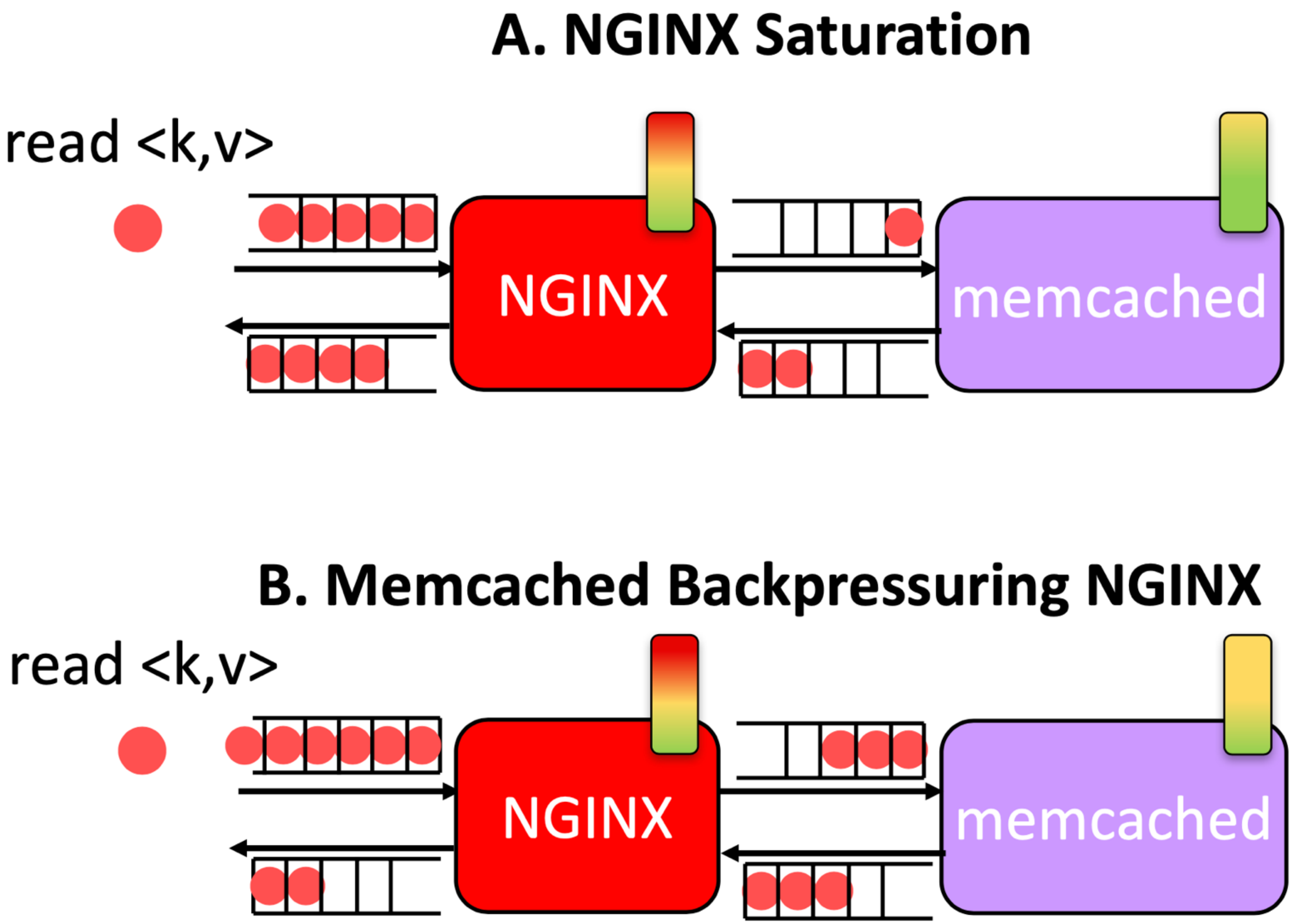


Figure 4: 22-27% of WSC cycles are spent in different components of “datacenter tax”.

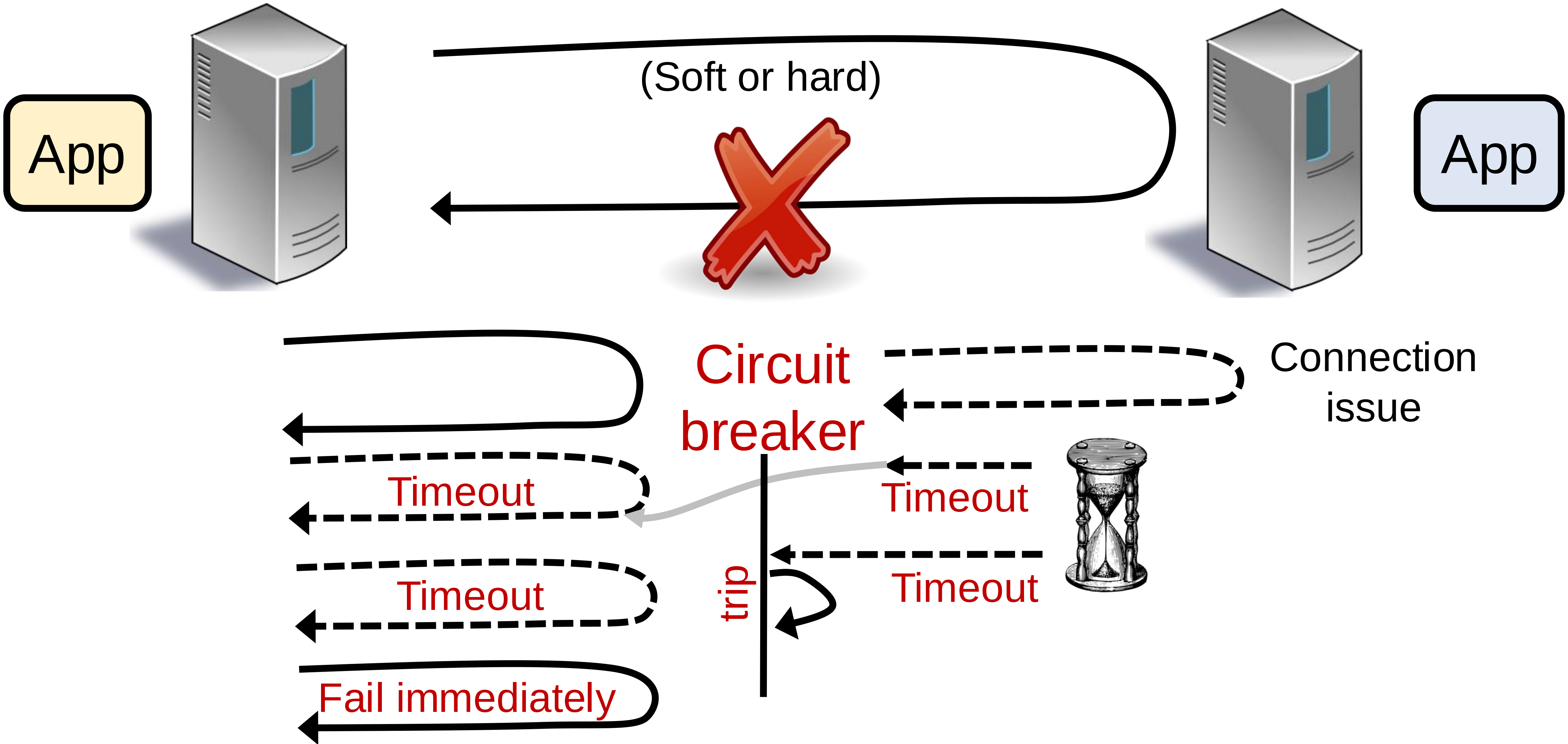
Profiling a warehouse-scale computer (Google). ISCA’15.

Cost of comm: Hotspot spreading



Deathstarbench. ASPLOS'19.

Cost of comm: high level failure handling



Are microservices always ideal?

- Just an architectural style. Look at solving problems first
- How to evolve the splitting of components?
  - Refactoring microservice interfaces later isn't easy
  - Interface changes need buy-in from multiple dev teams
  - Components should compose cleanly in the first place
- How to design apps?
  - Monolith first, or microservices from the beginning?
- Testing, Observability, Deploy automation
- How significant are dev coordination overheads?
- Complexity