Application Architecture

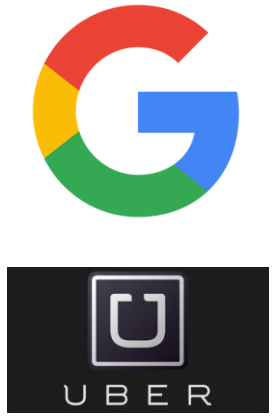
Lecture 5

Srinivas Narayana

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

RUTGERS
UNIVERSITY | NEW BRUNSWICK

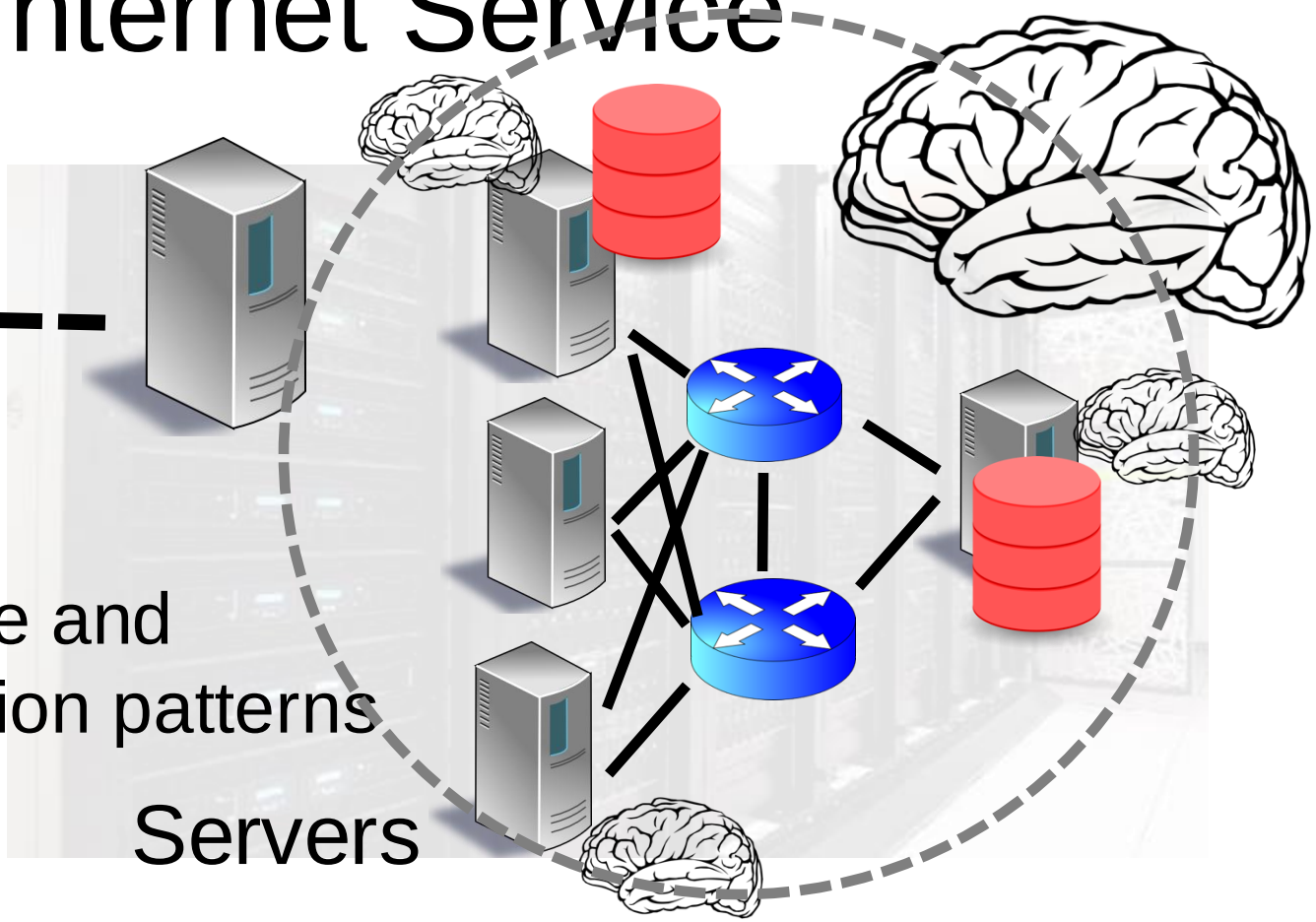
Components of an Internet Service



Routers

App compute and
communication patterns

Endpoints



Servers

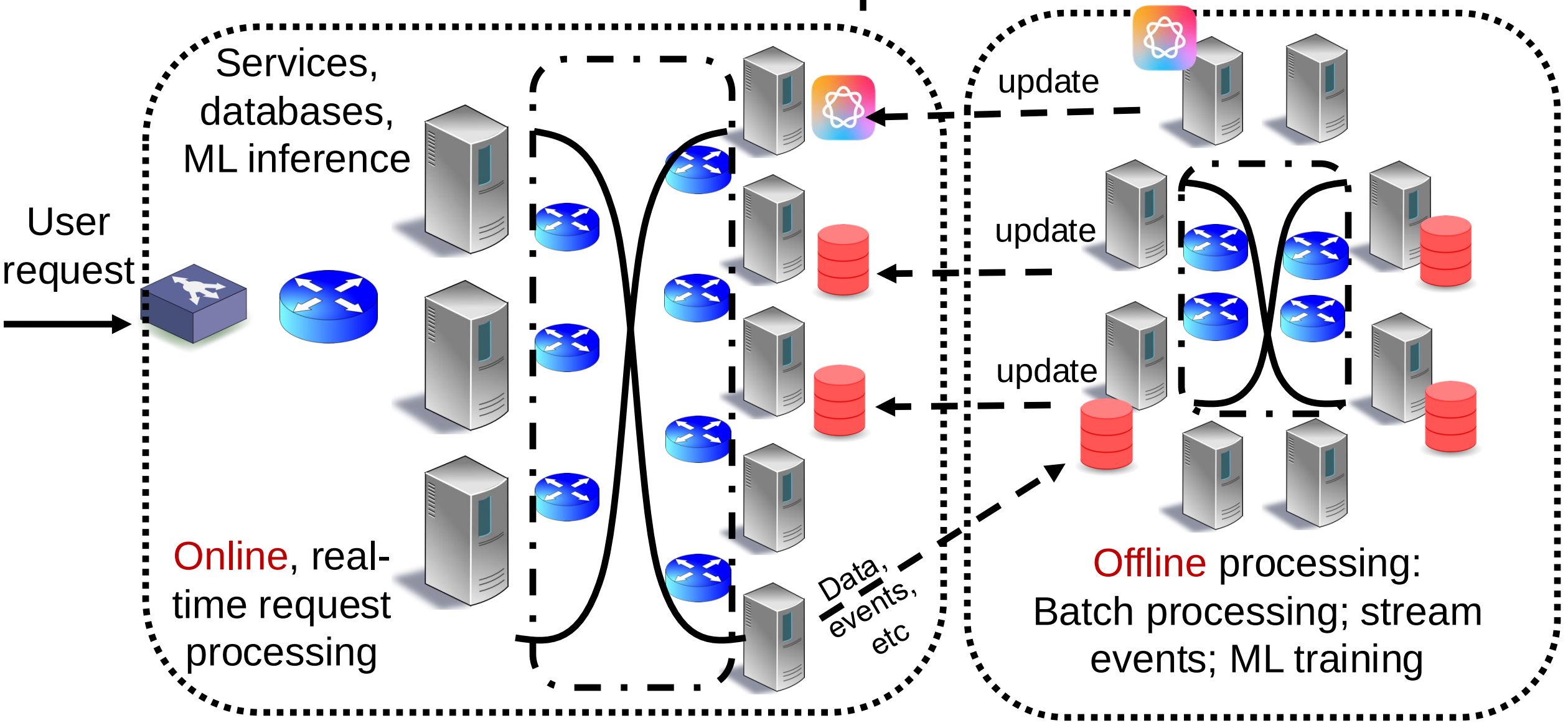
Modularized applications

Data Center

Storage

Interconnect: Routers

Offline and Online components



Review: Web server design

- Process other requests while waiting for one to finish

process

socket

IP_B + $port_B$

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

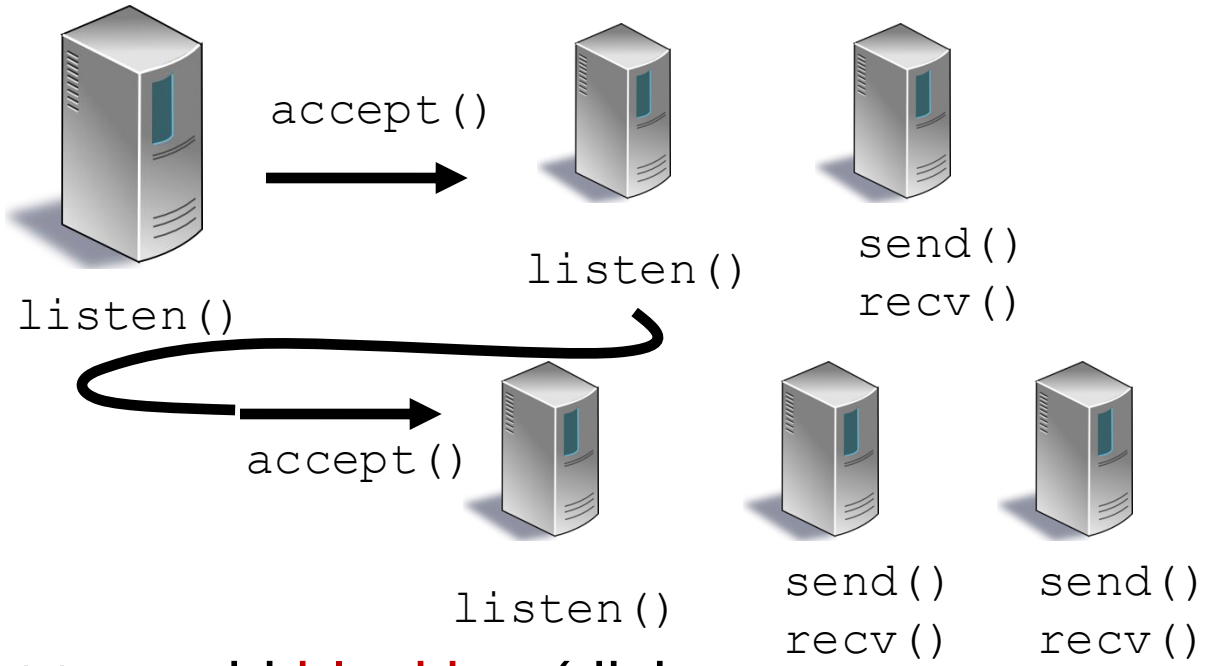
`listen()`

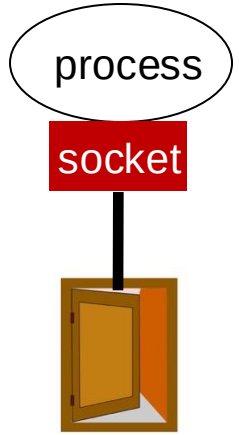
`accept()`

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

Review: 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()/..`

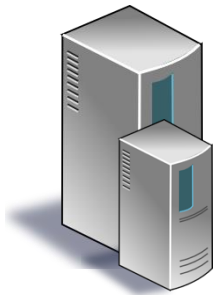
Great to avoid blocking (disk I/O, fastCGI, …)

Overhead grows with # connections ⟨ more / longer lived

Concurrency

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

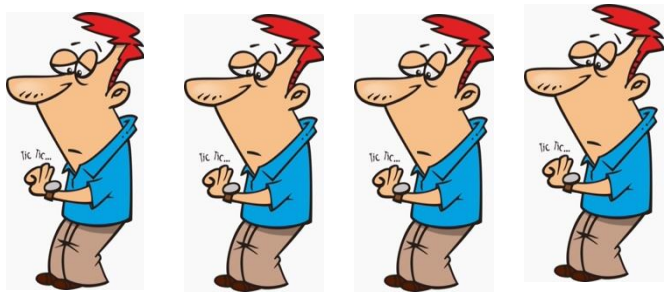
- Process other requests while waiting for one to finish
- A different design: single process event driven (SPED)



`accept()`

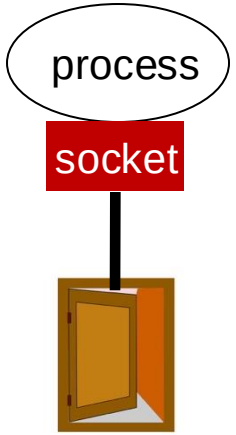
`recv()`

A queue of events



`epoll, select, kqueue, io_uring`

Lightweight



IP_B + $port_B$

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

`listen()`

`accept()`

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

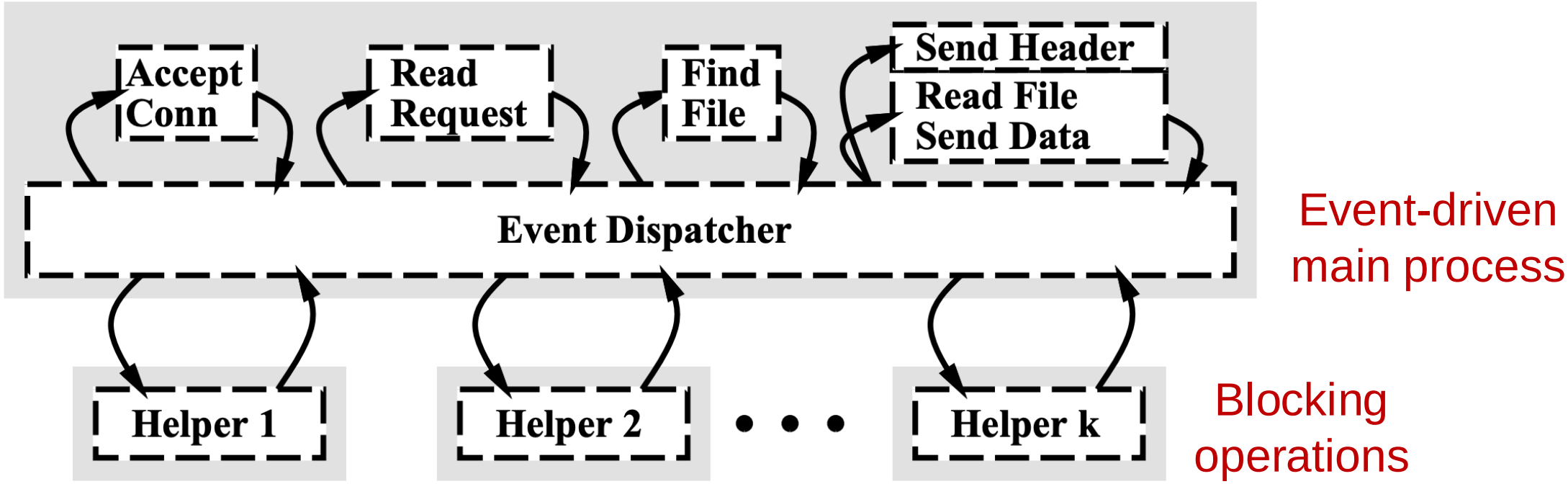
Can block if any of the requests block (asynchronous IO support can be incomplete & complex)

Avoid overheads of multiple processes and threads

https://aosabook.org/en/v2/nginx.html

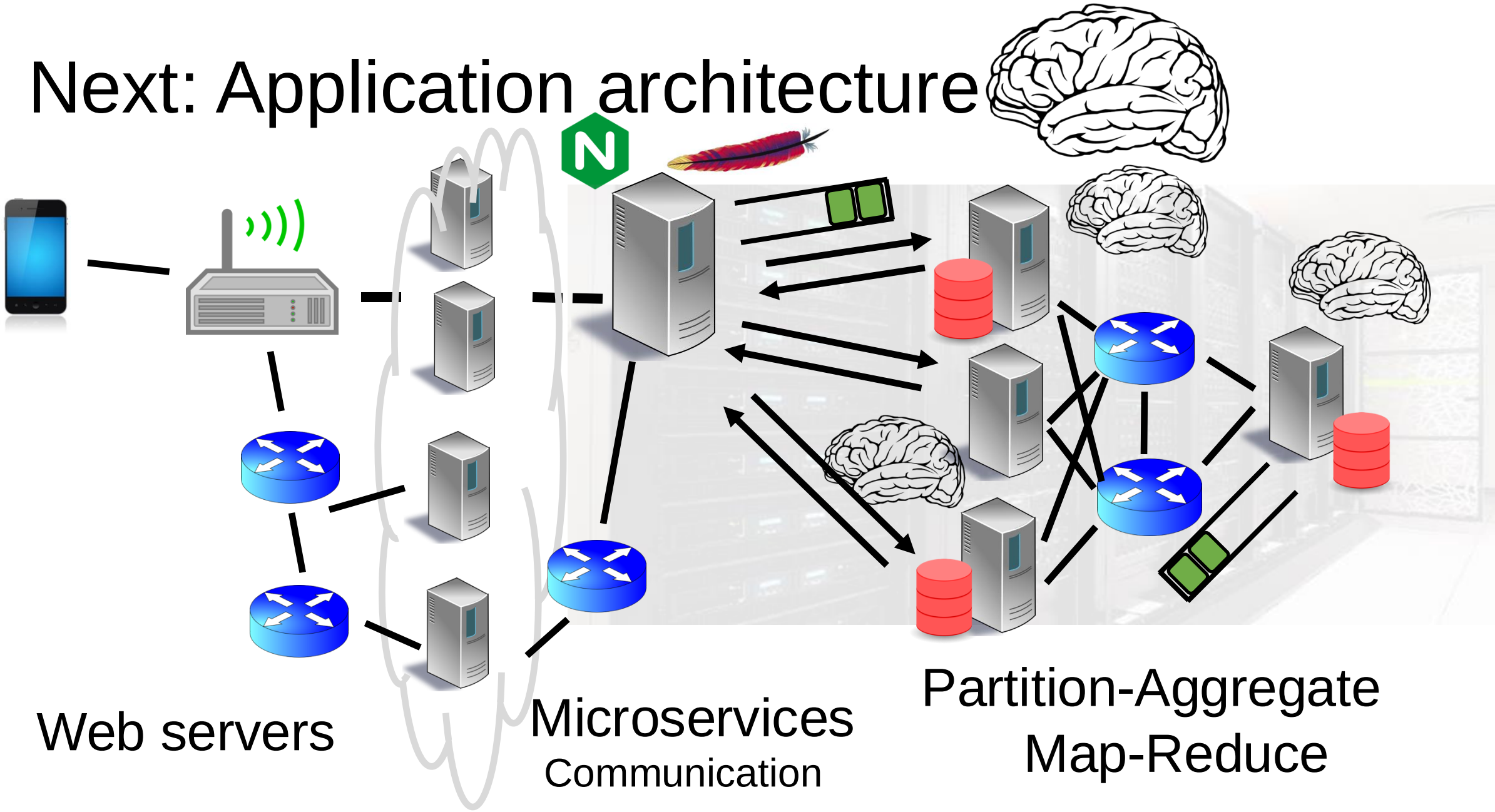
Using parallelism + concurrency

- Asymmetric Multi-Process Event Driven (AMPED)



Flash: An efficient and portable Web server

Next: Application architecture



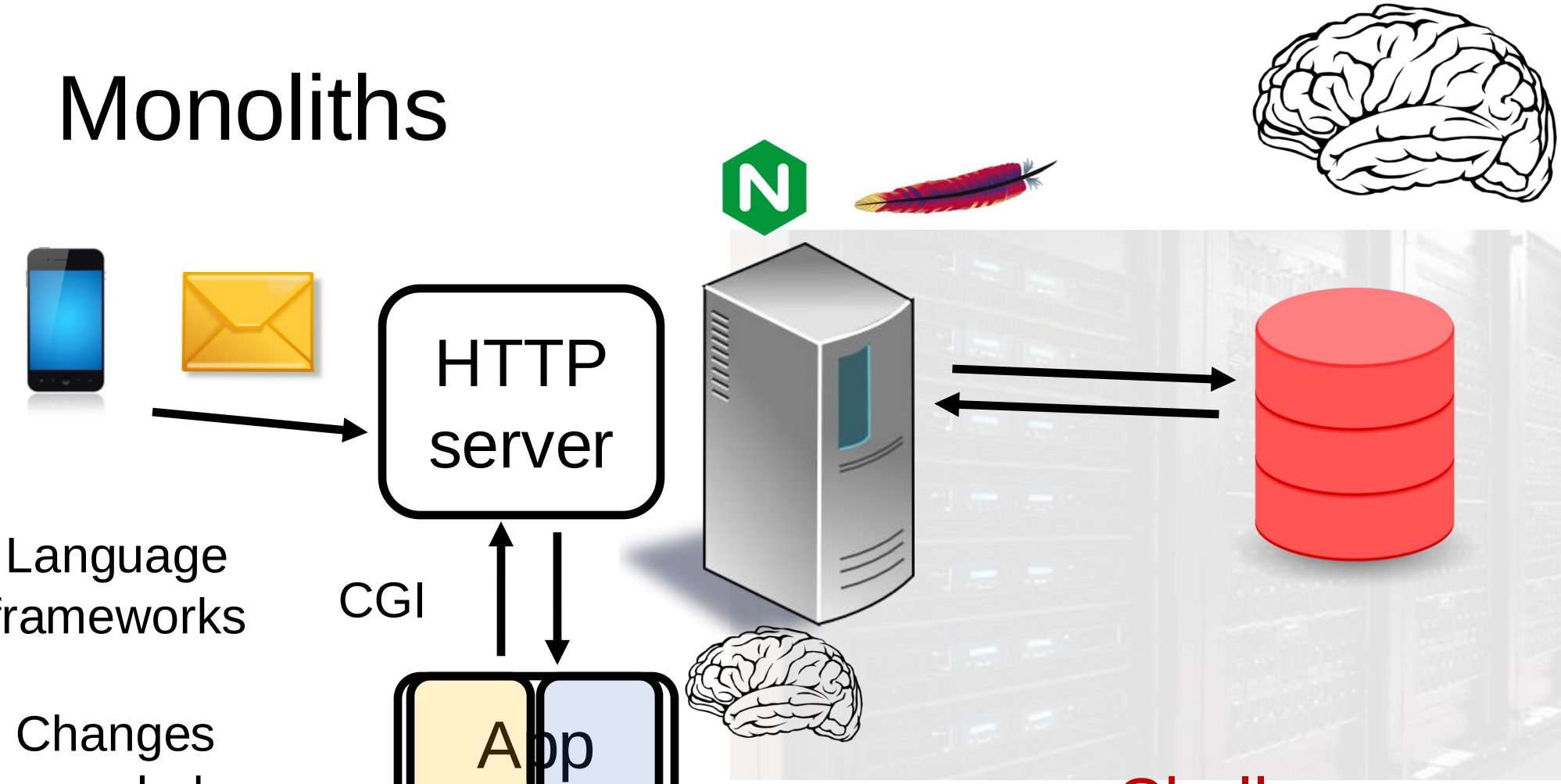
Web servers

Microservices
Communication

Partition-Aggregate
Map-Reduce

Microservice Architectural Pattern

Monoliths



Language frameworks

CGI

Changes coupled

Coordination across dev teams

Challenges

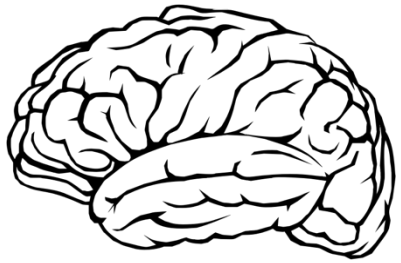
Releases

Transient functions

Scaling

Troubleshooting

Microservices



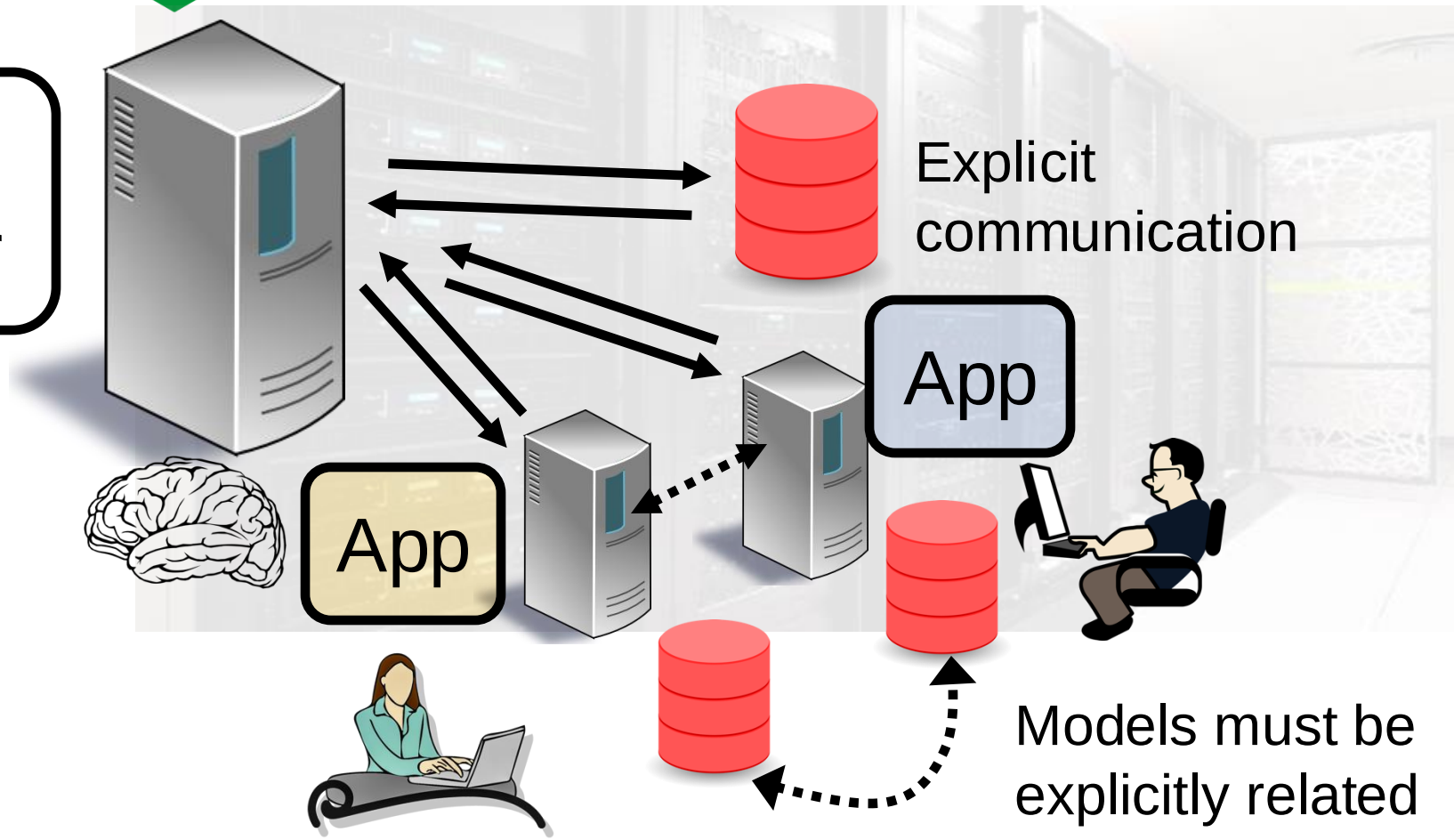
HTTP server

Language of choice

Independently upgrade and deploy

Independent/de-centralized data models & storage

Distinct from a software library: Out of process.



Loosely-coupled services

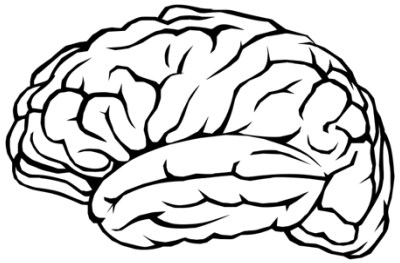
In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **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.

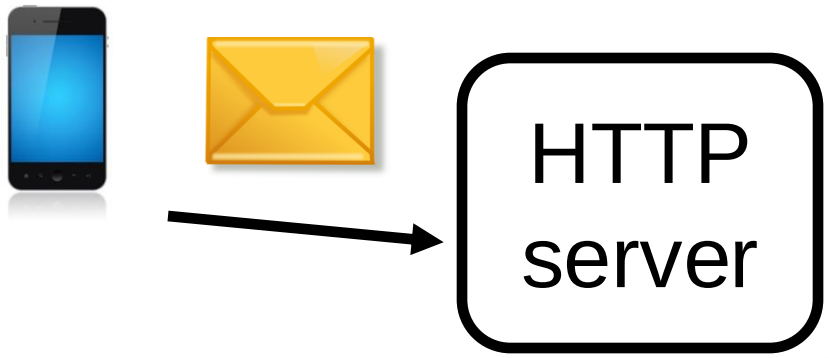
-- James Lewis and Martin Fowler (2014)

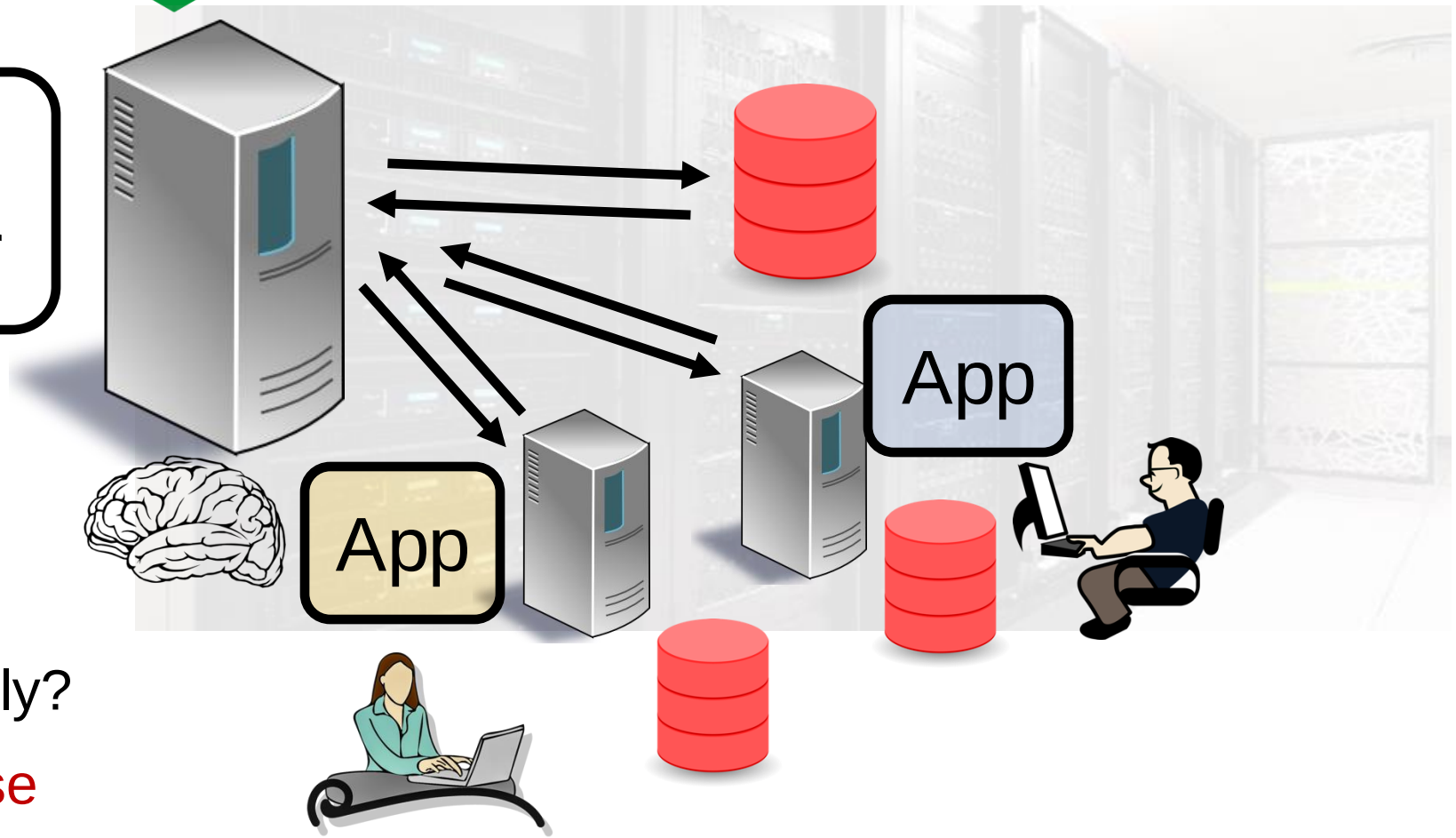
https://martinfowler.com/articles/microservices.html

https://martinfowler.com/microservices/

How to split?







Business Capabilities

Boundaries of change

Lifetime of the service?

Who should know (or not)?

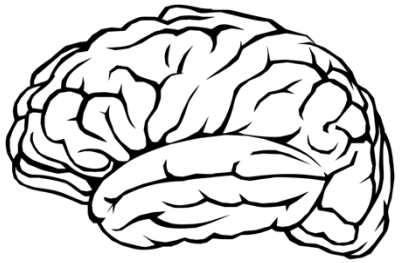
Changing together vs separately?

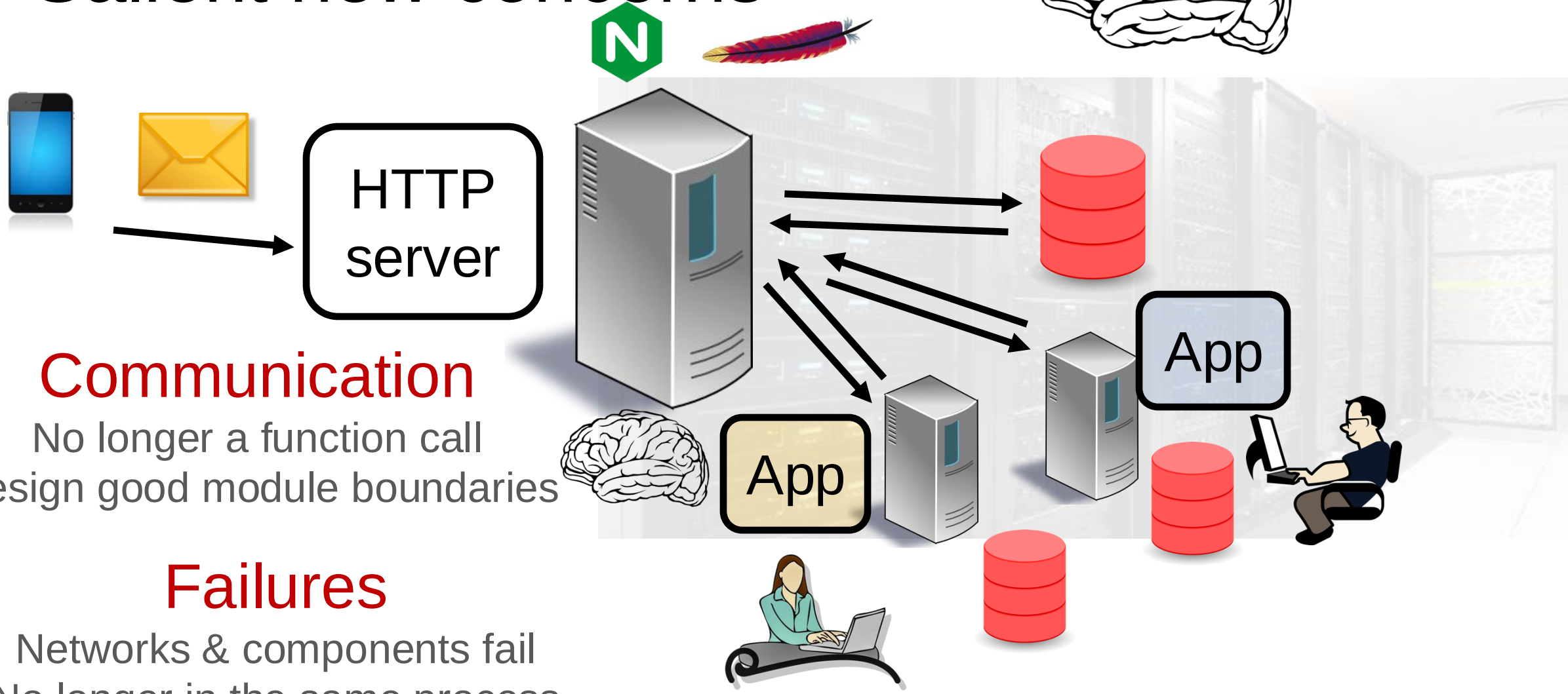
Heterogeneity in resource use

Refactoring common functionality

NETFLIX OSS

Salient new concerns





Communication

No longer a function call

Design good module boundaries

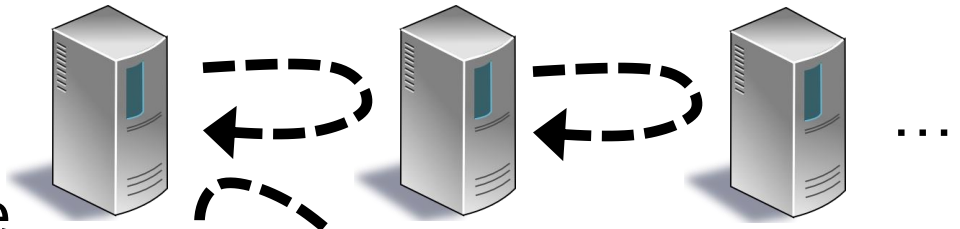
Failures

Networks & components fail

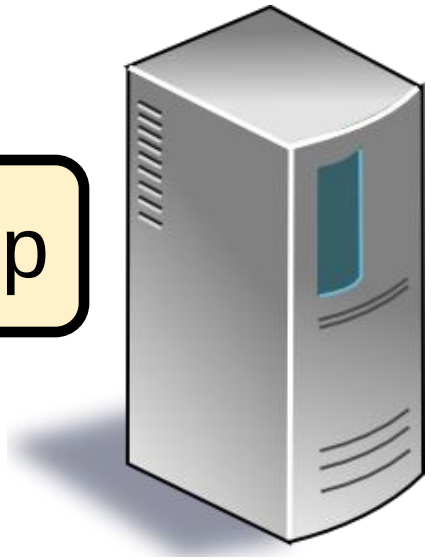
No longer in the same process

Communication

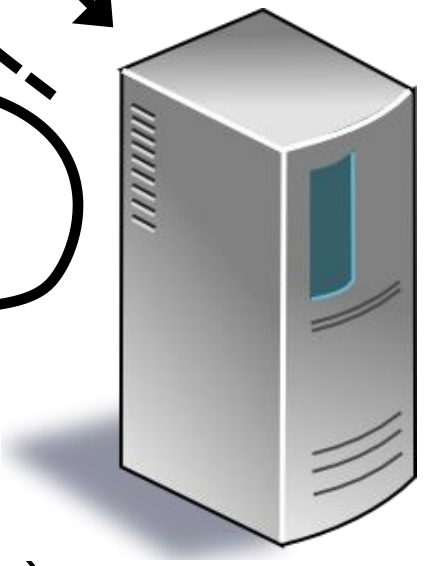
Lots of waiting;
Increasing
likelihood of failure



App



1 Synchronous blocking
(request-response)



App

Remote Procedure Call (RPC)

Serialization format (e.g., protobufs)

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

01101010101…
JSON
XML

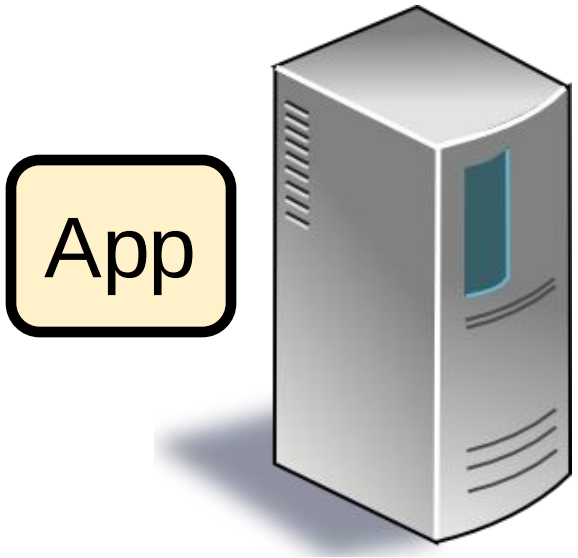
```
struct customer {
  string name;
  int customer_id;
  ..;
}
```

Communication

4 Shared data (async)

3 Event streaming (asynchronous)

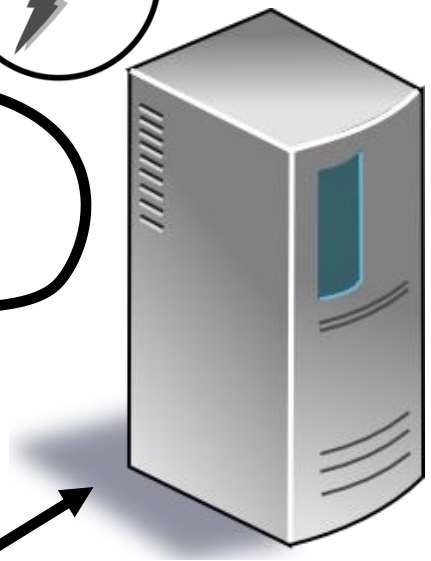
Not expecting a response



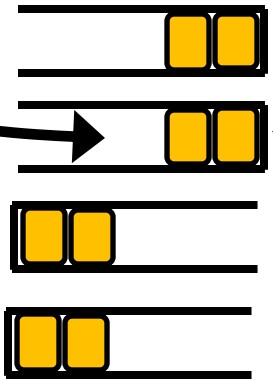
1 Synchronous blocking (request-response)

2 Async Qs

Message router



Can do useful work while waiting (but still need timeouts)



2 Asynchronous request-response

ØMQ

`callback()`

"Building microservices", Sam Newman

Cost of communication: Performance

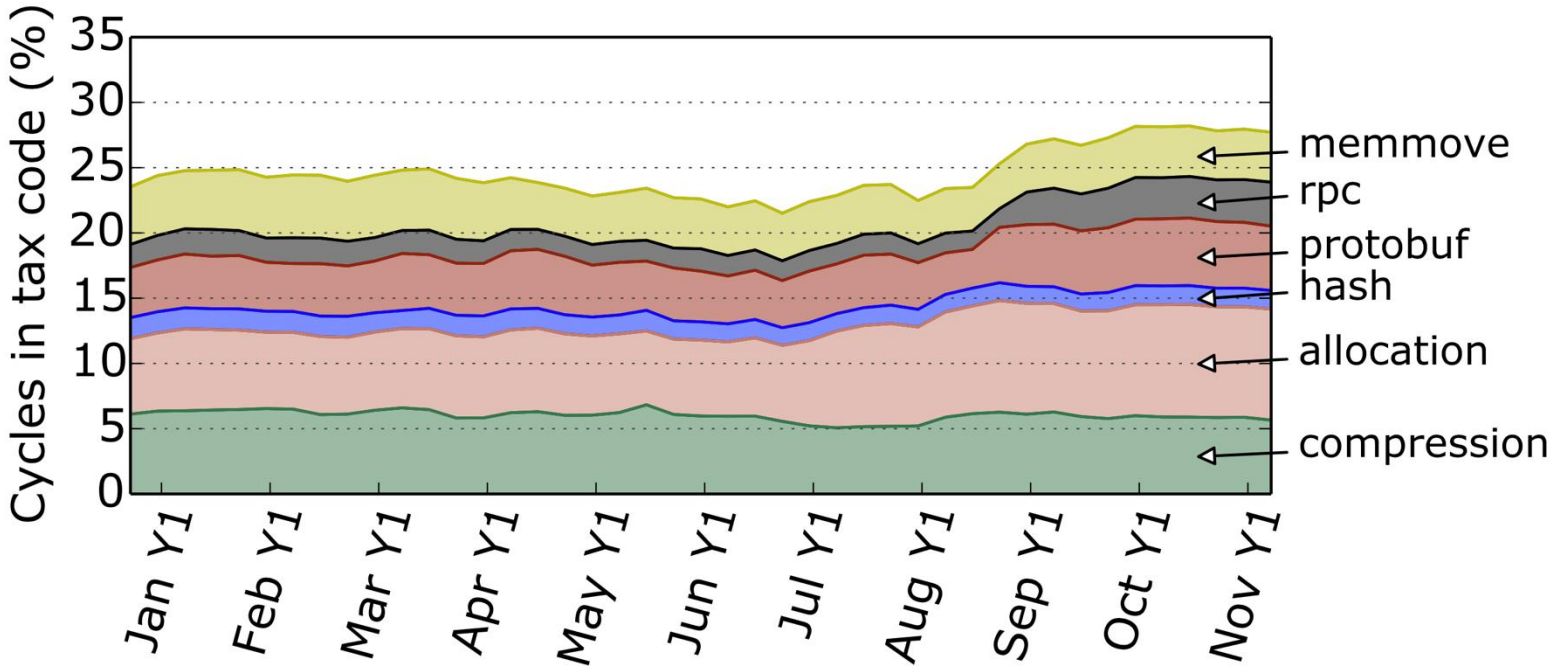
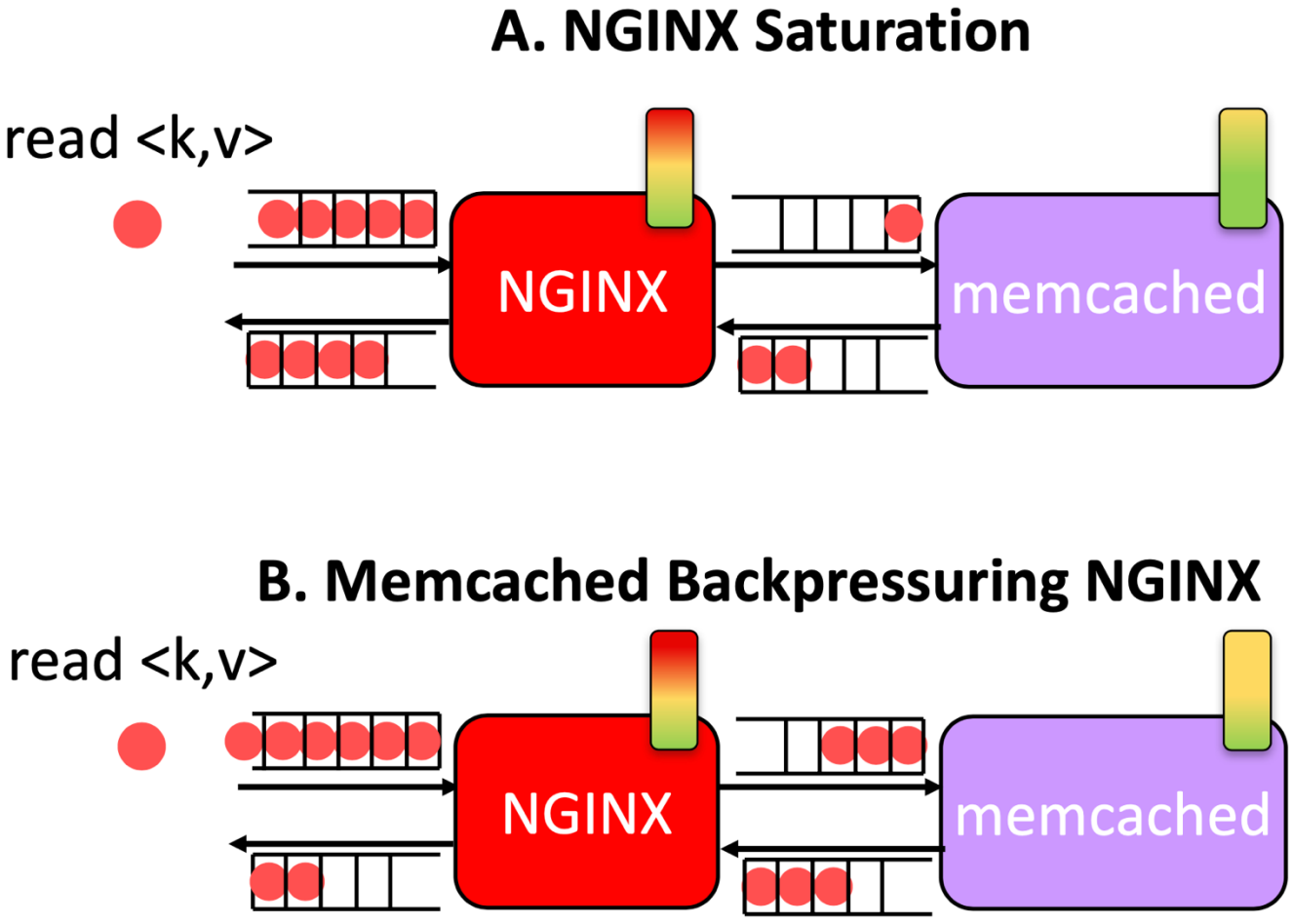


Figure 4: 22-27% of WSC cycles are spent in different components of "datacenter tax".

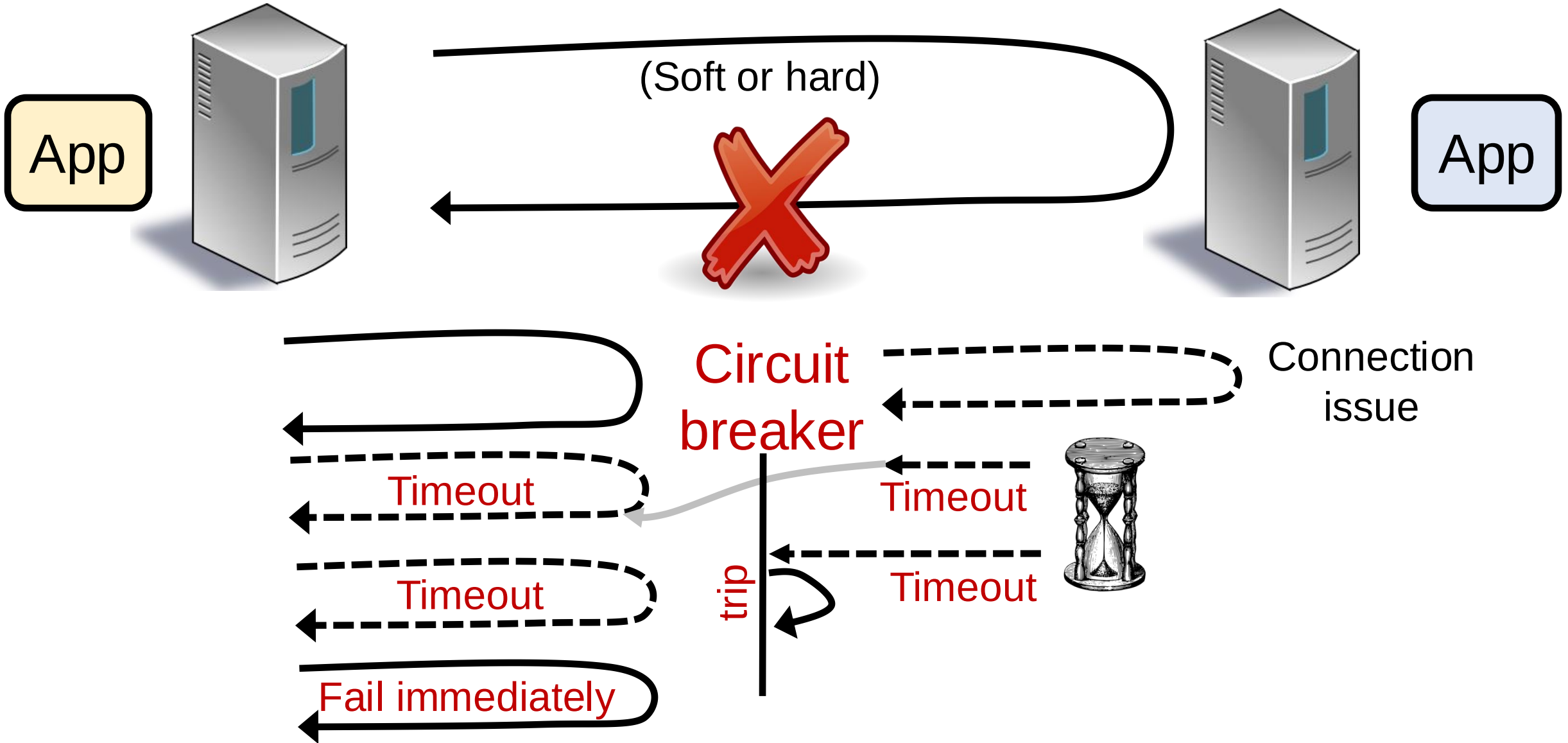
Profiling a warehouse-scale computer (Google). ISCA'15.

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

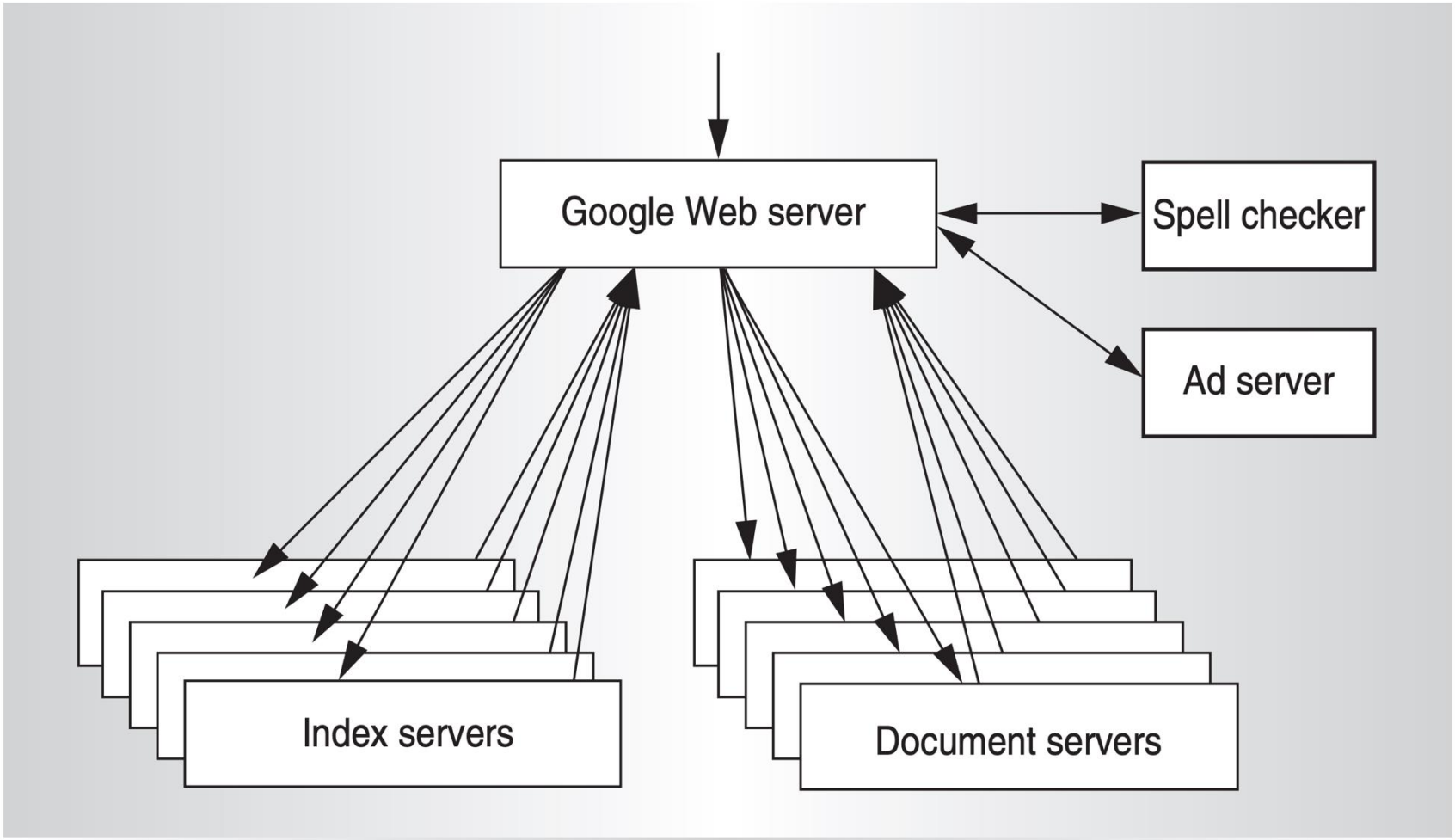
Partition-Aggregate

Processing interactive search queries

Web search: some numbers (circa 2003)

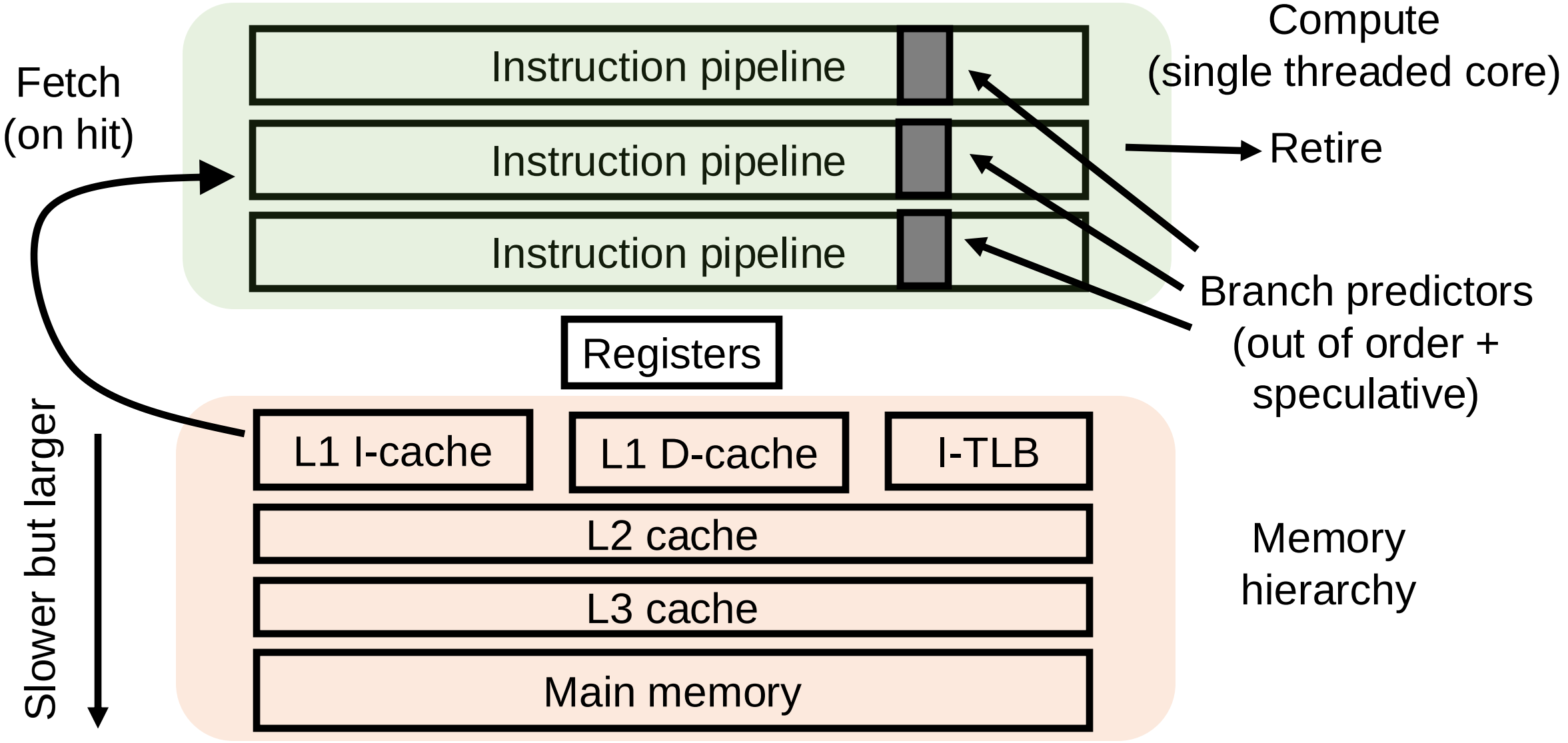
- 10s of terabytes of web corpus data
  - Read 100s of megabytes per query
- 10s of billions of CPU instructions per query
- Data accessed depends on the query; hard to predict
- All results to be returned to users within (say) 300 milliseconds
- Cannot process on a single machine within acceptable time

Example: Google search architecture



Web search for a planet, MICRO'03

Quick Review: Compute & Memory Org



Measurements from one (index) server

- Not too fast single-threaded
  - Data dependencies
  - Branches often mispredicted
- Small instruction memory footprint
- Data locality within a block, but not across blocks
- Numbers not much better on a newer architecture
- Can't drive high single-threaded performance

Use parallelism

Characteristic	Value
Cycles per instruction	1.1
Ratios (percentage)	
Branch mispredict	5.0
Level 1 instruction miss*	0.4
Level 1 data miss*	0.7
Level 2 miss*	0.3
Instruction TLB miss*	0.04
Data TLB miss*	0.7

* Cache and TLB ratios are per instructions retired.

Web search for a planet, MICRO'03.