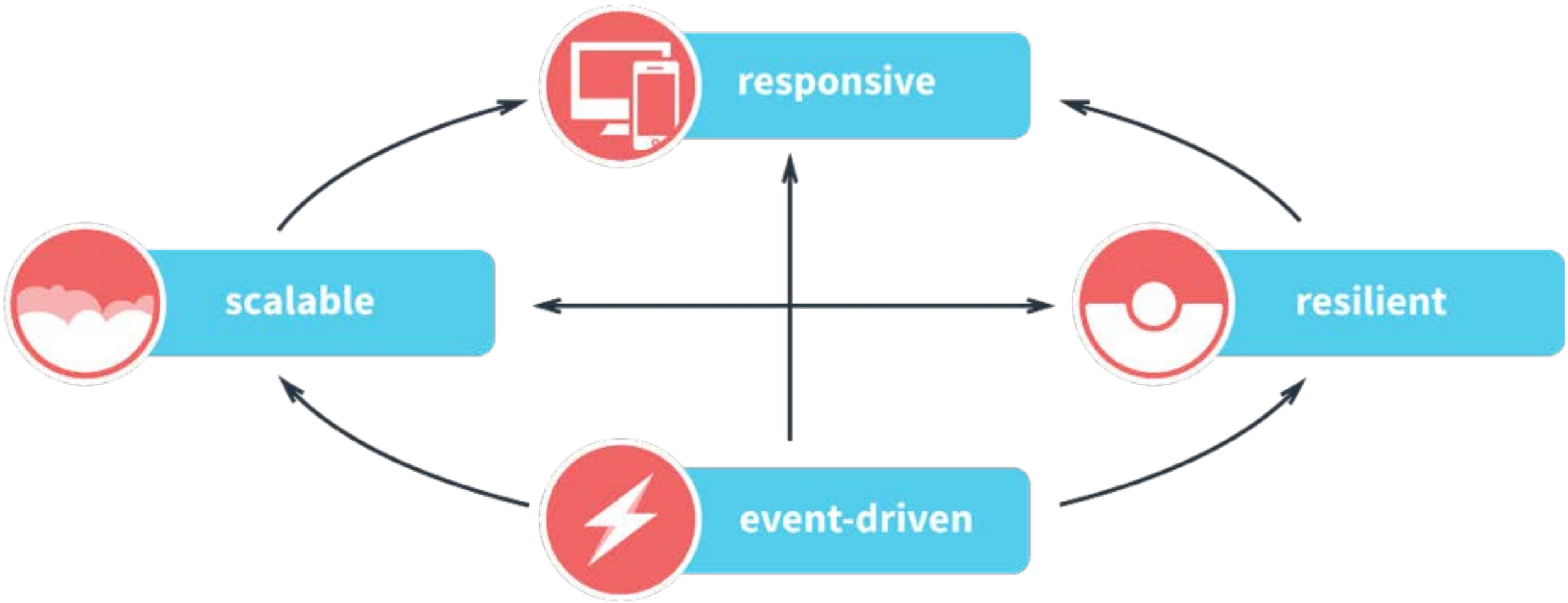
Go Reactive

Blue Print for Future Applications

Dr. Roland Kuhn
Akka Tech Lead
@rolandkuhn

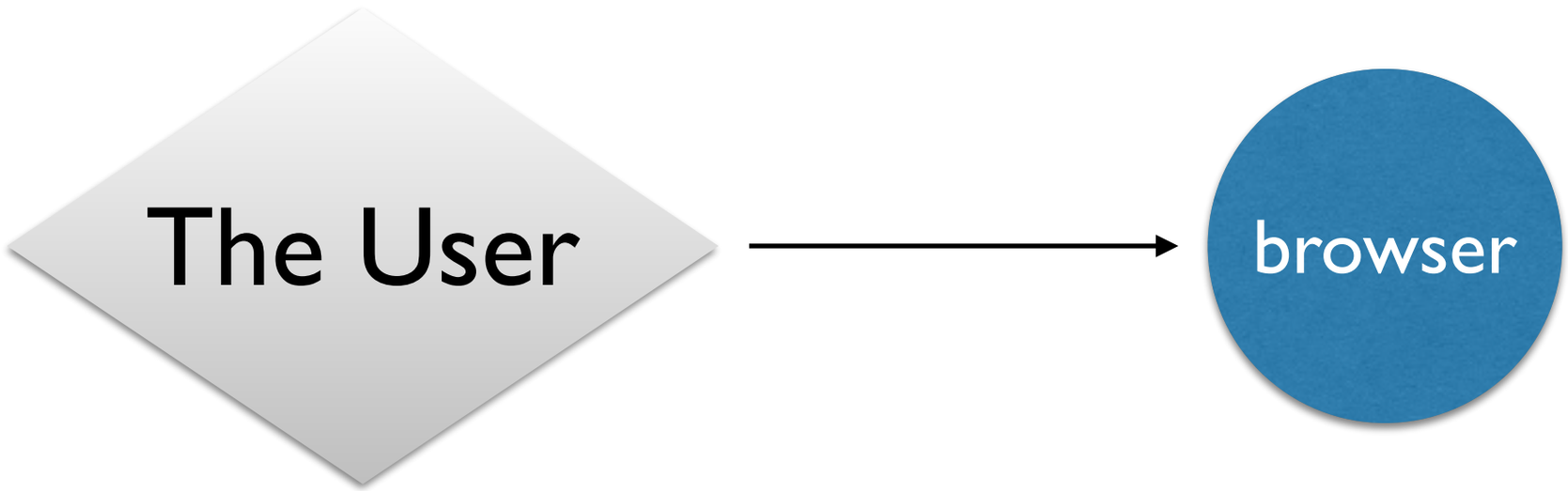
The Four Reactive Traits

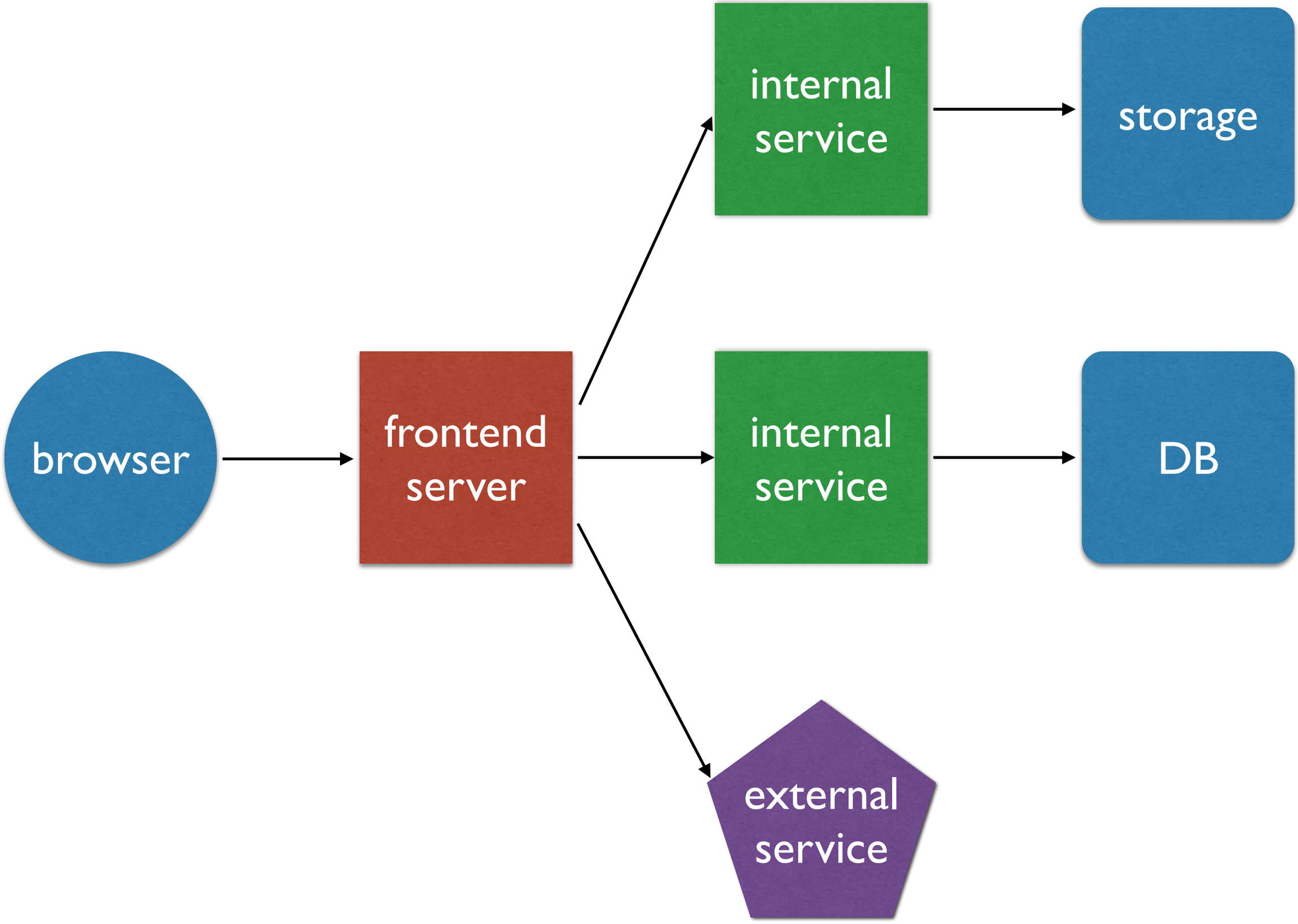


http://reactivemanifesto.org/

Starting Point:

The User


The User
browser


internal service
storage
browser
frontend server
internal service
DB
external service

Responsiveness

always available
interactive
(near) real-time

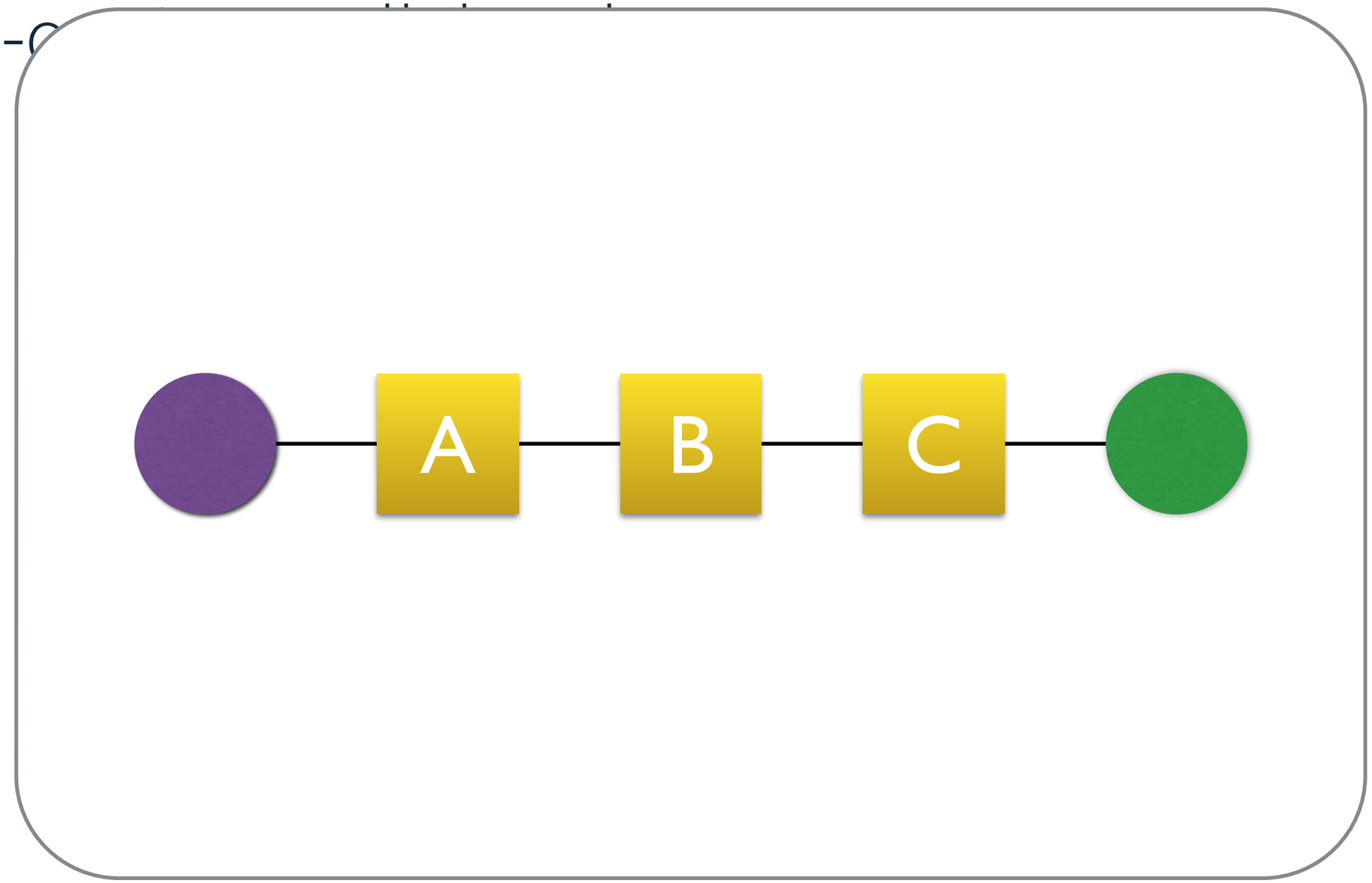
Bounded Latency

Bounded Latency

- fan-out in parallel and aggregate

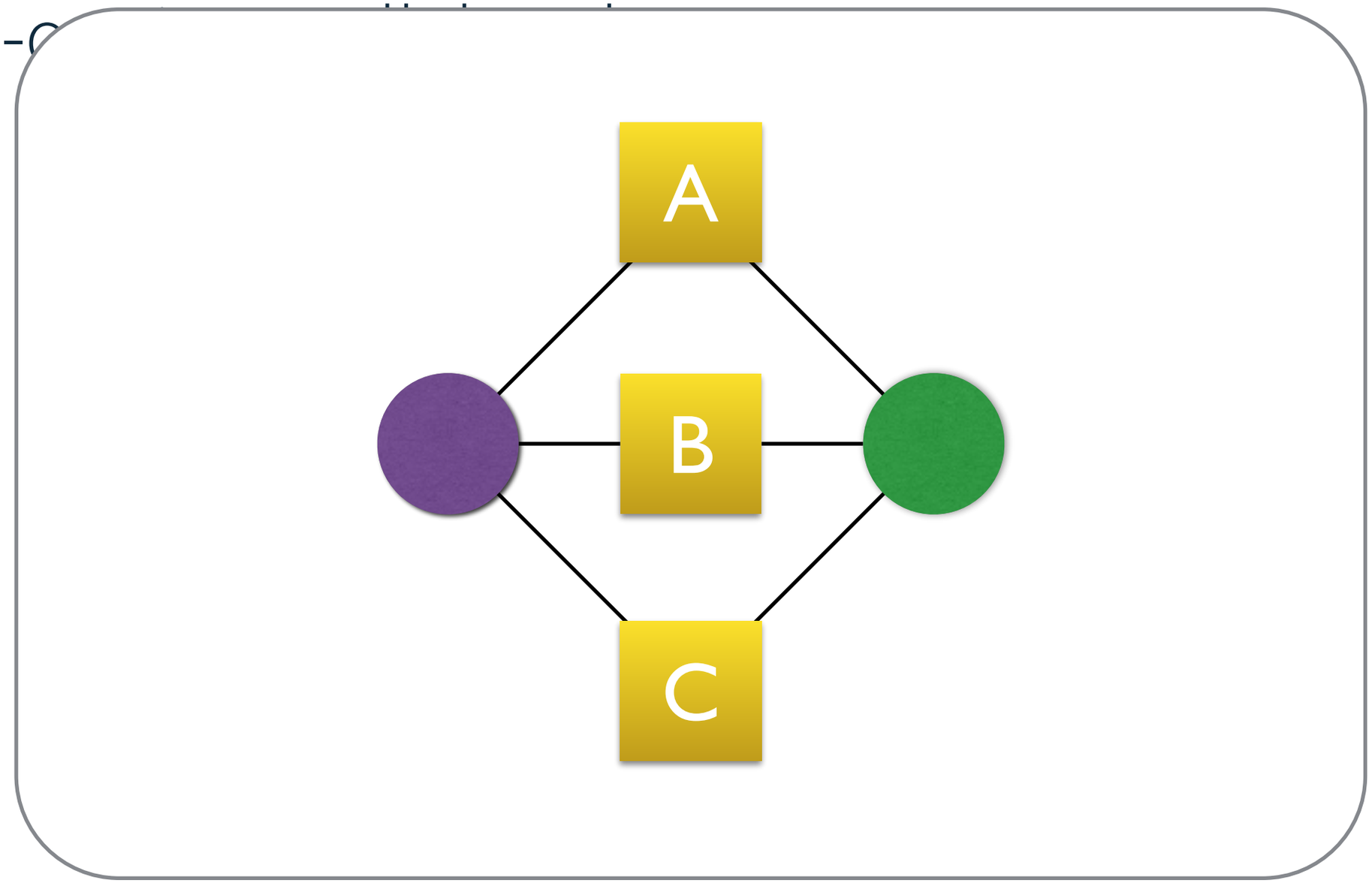
Bounded Latency

- fan-o



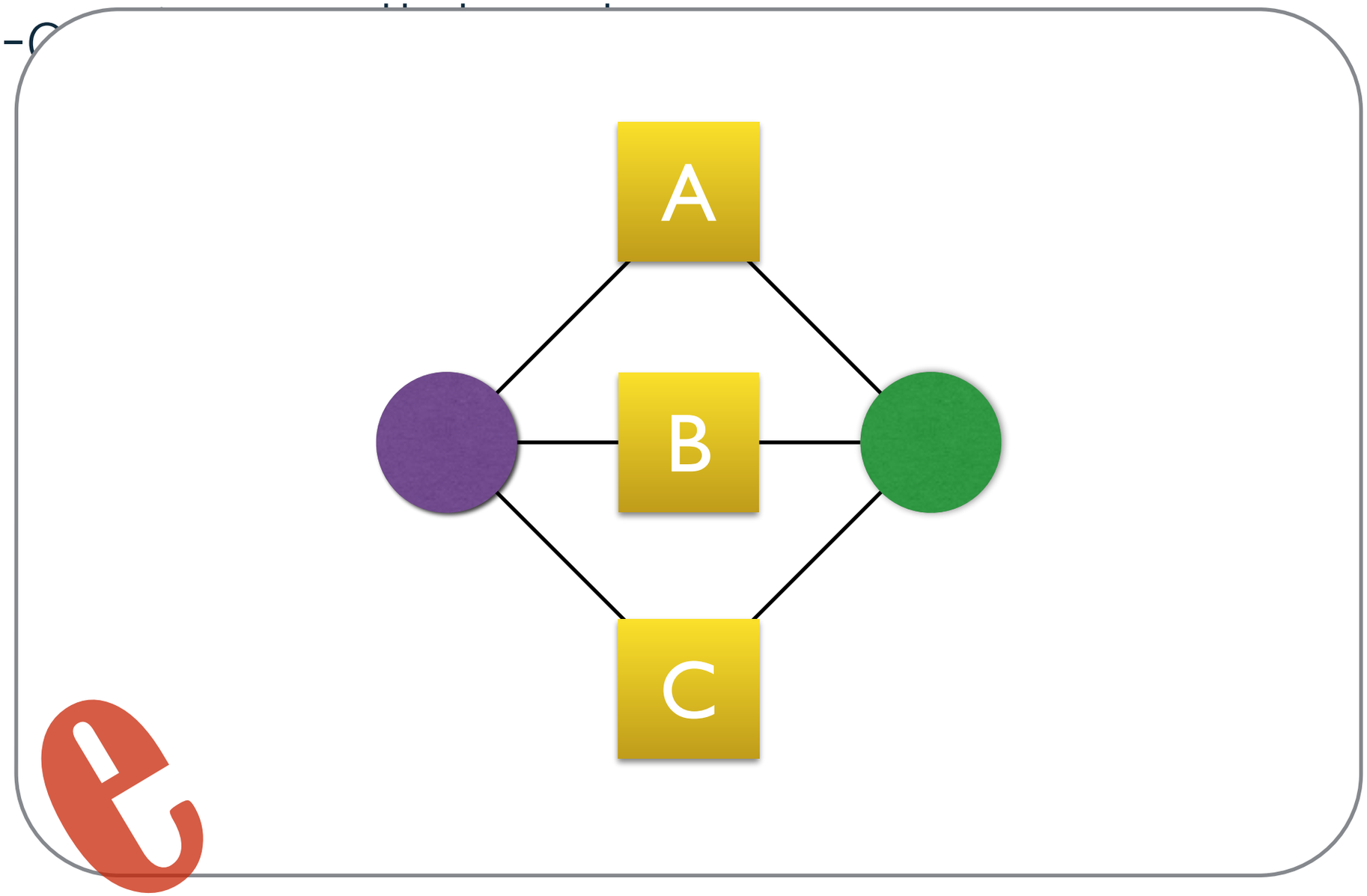
Bounded Latency

- fan-o



Bounded Latency

- fan-o

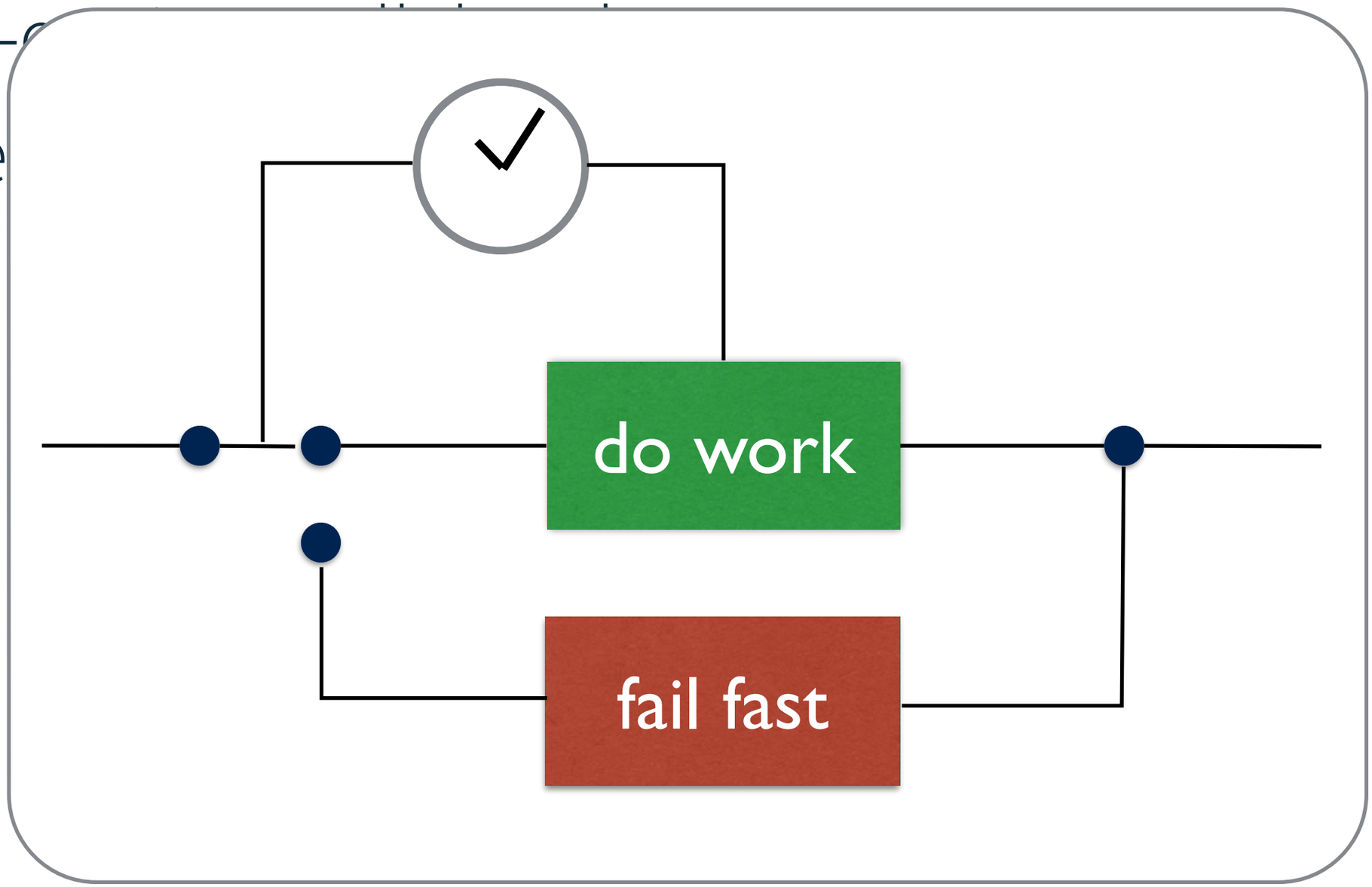


Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation

Bounded Latency

- fan-
- use



Bounded Latency

- fan-out in parallel and aggregate
- use circuit breakers for graceful degradation
- use bounded queues, measure flow rates

Bounded Latency

- fan-
- use
- use

Use Bounded Queues:

Latency = QueueLength • ProcessingTime

(for reasonably stable average processing time)

Bounded Latency

- fan-
- use
- use

Use Bounded Queues:

Latency = QueueLength • ProcessingTime

(for reasonably stable average processing time)

Resilience

Responsive in the Face of **Failure**

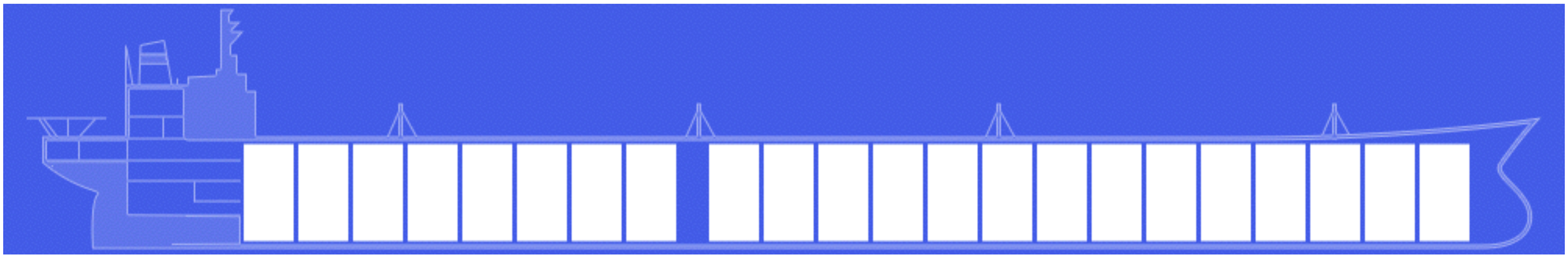
Handle Failure

- software will fail
- hardware will fail
- humans will fail
- system still needs to respond ➟ **resilience**

Distribute!

Asynchronous Failure

- parallel fan-out & distribution
  ➠ asynchronous execution
- compartmentalization & isolation

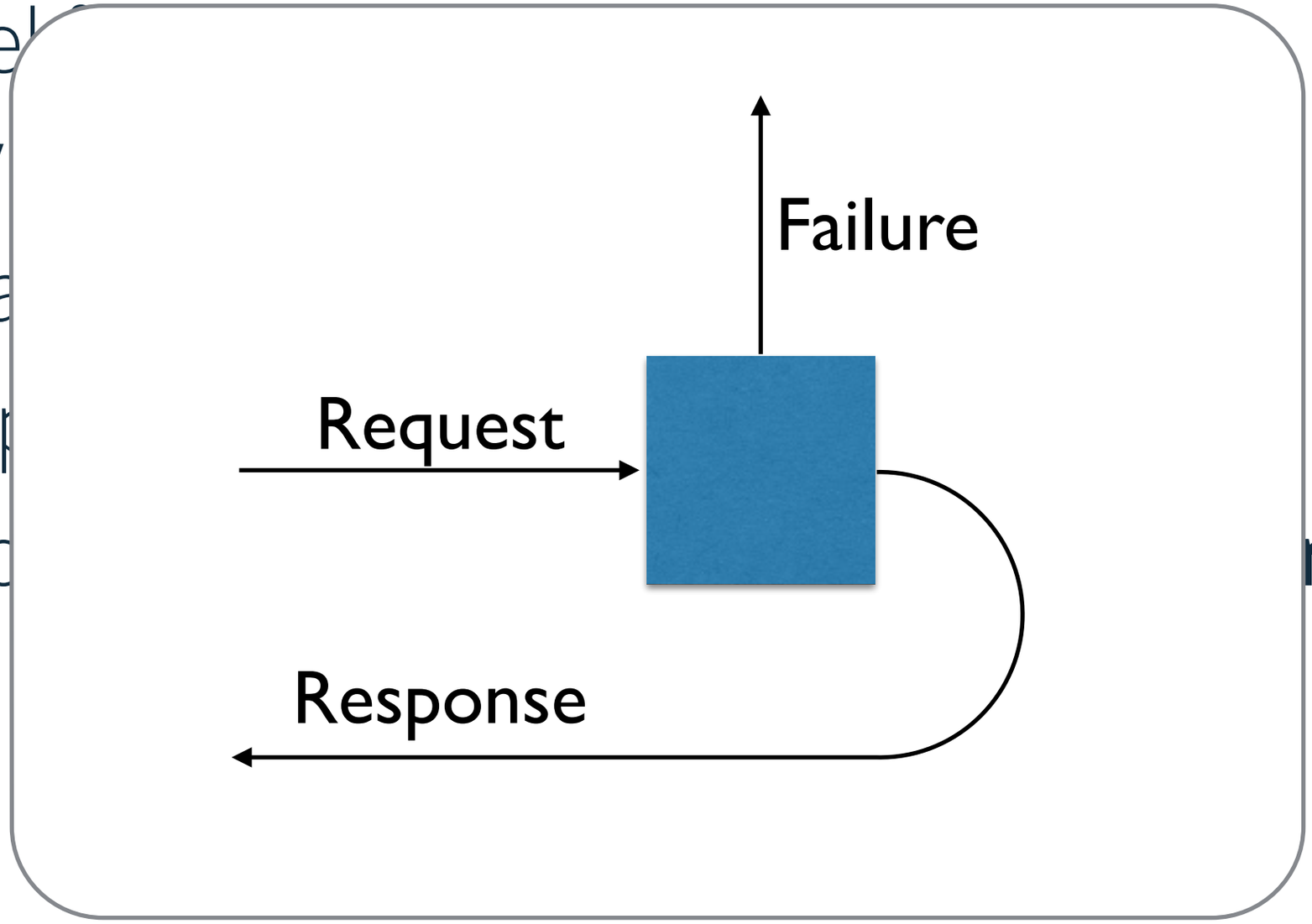


Asynchronous Failure

- parallel fan-out & distribution
  ➟ asynchronous execution
- compartmentalization & isolation
- no response? ➟ timeout events
- **s**omeone **e**lse's exce**p**tion? ➟ **supervision**

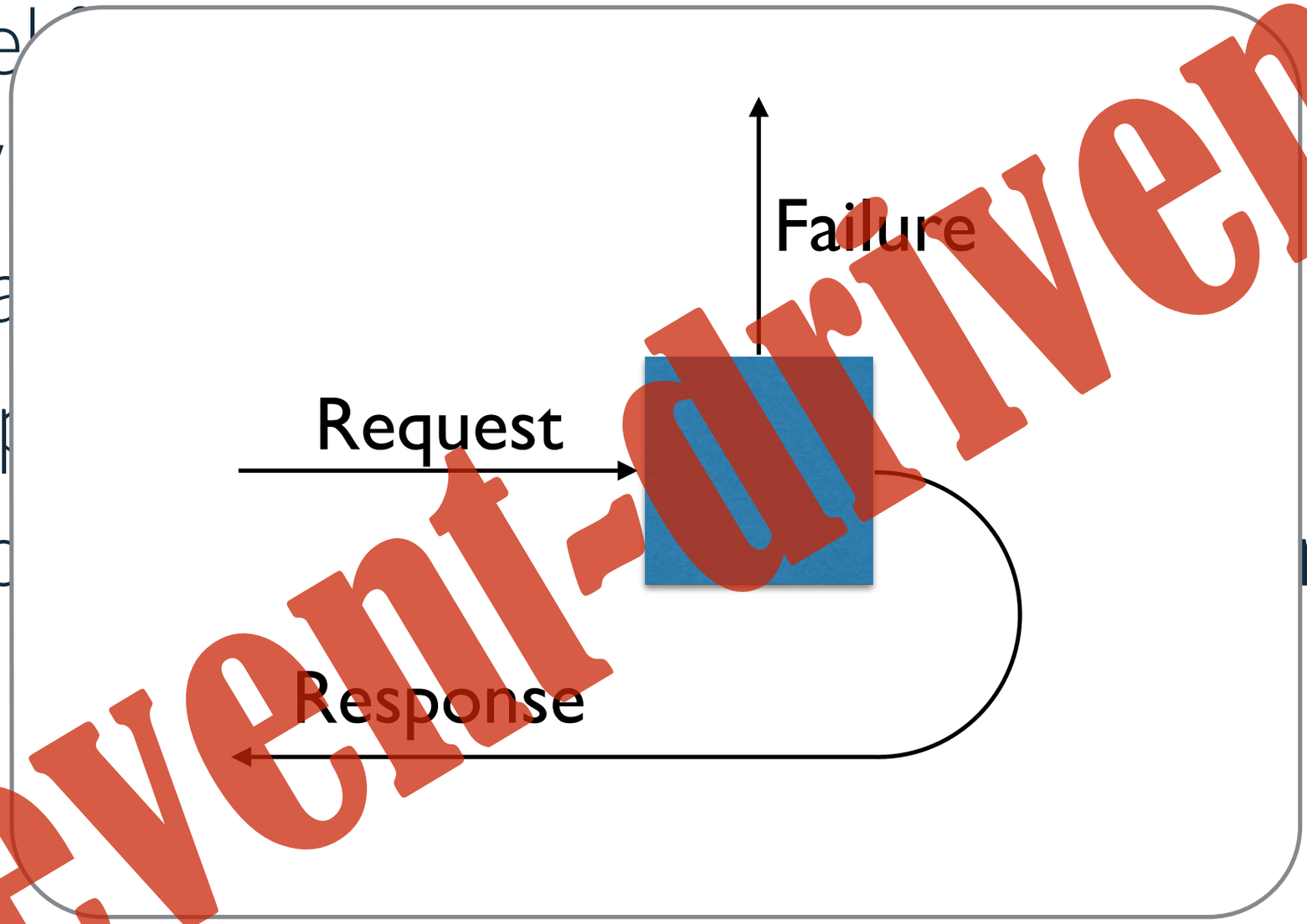
Asynchronous Failure

- parallel
  - ➠ asy
- compa
- no resp
- someo



Asynchronous Failure

- parallel
  - ➠ asy
- compa
- no resp
- **s**omec **n**



Asynchronous Failure

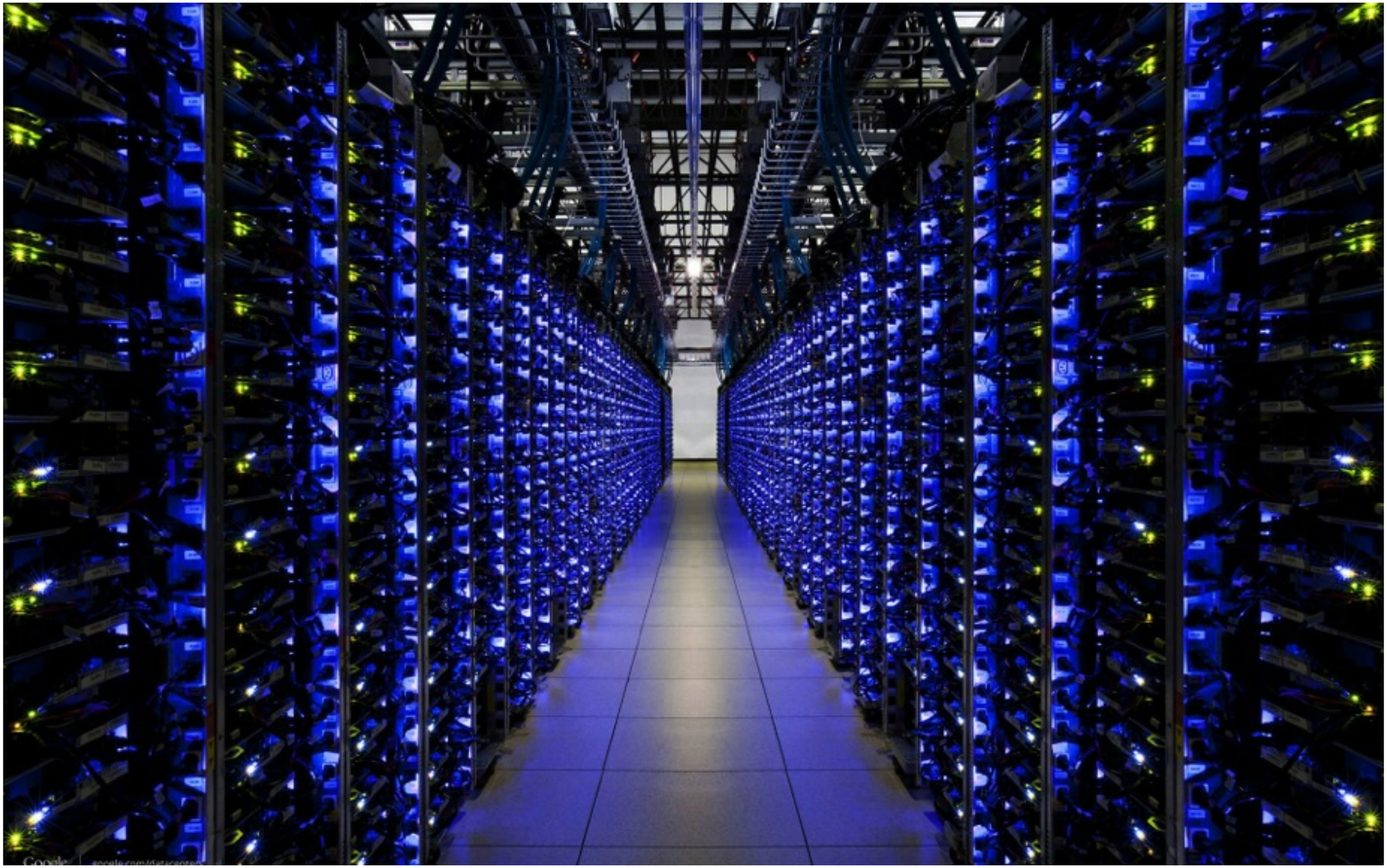
- parallel fan-out & distribution
  ➟ asynchronous execution
- compartmentalization & isolation
- no response? ➟ timeout events
- **s**omeone **e**lse's exce**p**tion? ➟ **supervision**
- **location transparency** ➟ seamless resilience

Scalability

Responsive in the Face of **Changing Load**

Handle Load

Handle Load



Handle Load

- partition incoming work for distribution
- share nothing
- scale capacity up and down on demand
- supervise and adapt
- **location transparency**
  - ➟ seamless scalability

Handle Load

- partition incoming work for distribution
- share nothing
- scale capacity up and down on demand
- supervise and adapt
- **location transparency**
  - ➟ seamless scalability

… this has some interesting consequences!

Consequences

- distribution & scalability
  - ➟ loss of strong consistency
- CAP theorem? — not as relevant as you think
- eventual consistency
  - ➟ gossip, heartbeats, dissemination of change

Pat Helland: Life beyond Distributed Transactions

Peter Bailis: Probabilistically Bounded Staleness (http://pbs.cs.berkeley.edu)

Consequences

- distribution & scalability
  - ➟ loss of strong consistency
- CAP theorem? — not as relevant as you think
- eventual consistency
  - ➟ gossip, heartbeats, dissemination of change

Pat Helland: Life beyond Distributed Transactions

Peter Bailis: Probabilistically Bounded Staleness (http://pbs.cs.berkeley.edu)

Corollary

- Reactive needs to be applied all the way down
- Polyglot deployments demand collaboration
  ➟ for example Reactive Streams

But what about us,
the developers?

Step 1: Take a Leap of Faith

- thread-based models have made us defensive
  - “don’t let go of your thread!”
  - “asynchrony is suspicious”
  - “better return strict value, even if that needs blocking”

Step 1: Take a Leap of Faith

- thread-based models have made us defensive
  - “don’t let go of your thread!”
  - “asynchrony is suspicious”
  - “better return strict value, even if that needs blocking”
- **it is okay to write a method that returns a Future!**

Step 2: Rethink the Architecture

- break out of the synchronous blocking prison
- focus on communication & protocols
- asynchronous program flow
  - ➟ no step-through debugging
  - ➟ tracing and monitoring
- **loose coupling**

Step 3: Profit!

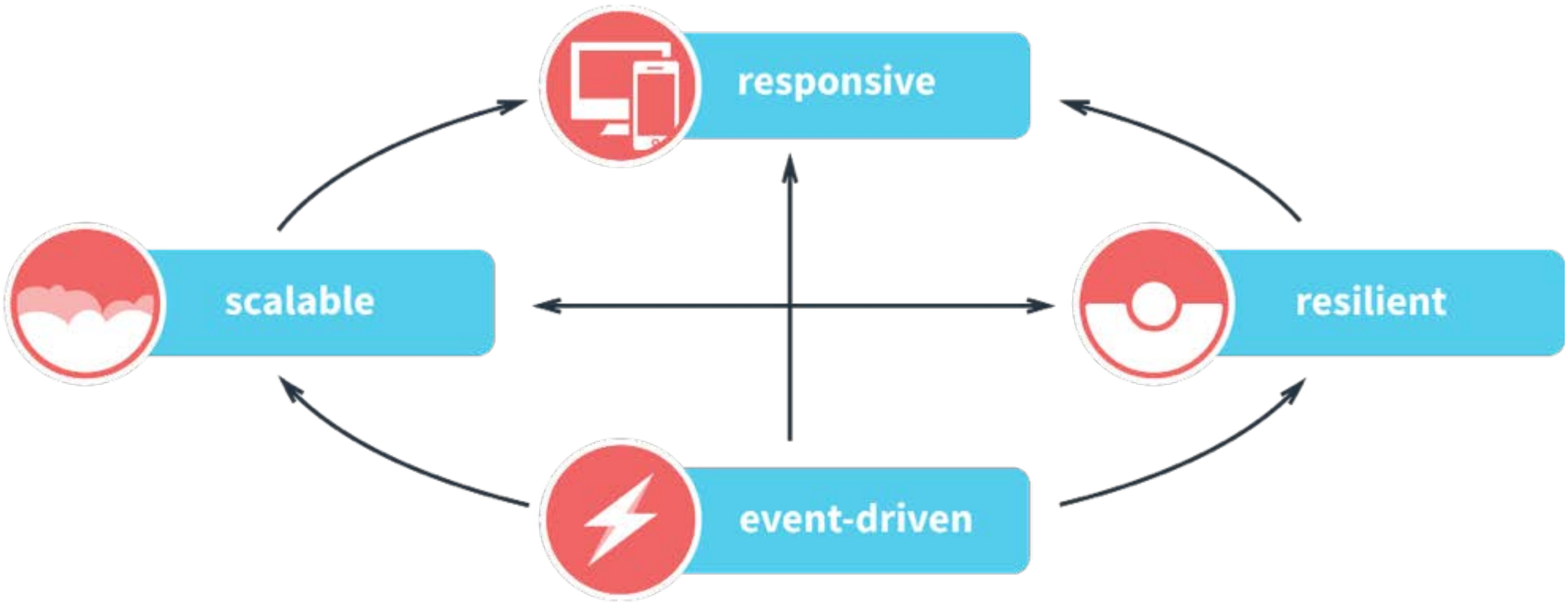
- clean business logic, separate from failure handling
- distributable units of work
- effortless parallelization
- less assumptions ➟ lower maintenance cost

Step 3: Profit!

- clean business logic, separate from failure handling
- distributable units of work
- effortless parallelization
- less assumptions ➟ lower maintenance cost
- independent agents ➟ **fun to work with!**

Summary

The Four Reactive Traits



http://reactivemanifesto.org/