Concurrency + Distribution = Availability + Scalability

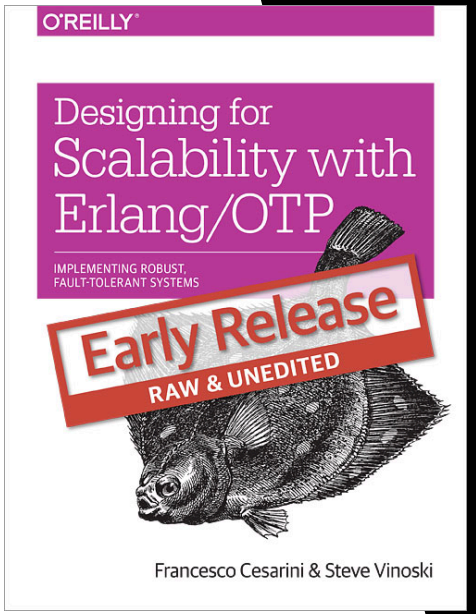
Francesco Cesarini

francesco@erlang-solutions.com
www.erlang-solutions.com
@francescoC

Erlang SOLUTIONS

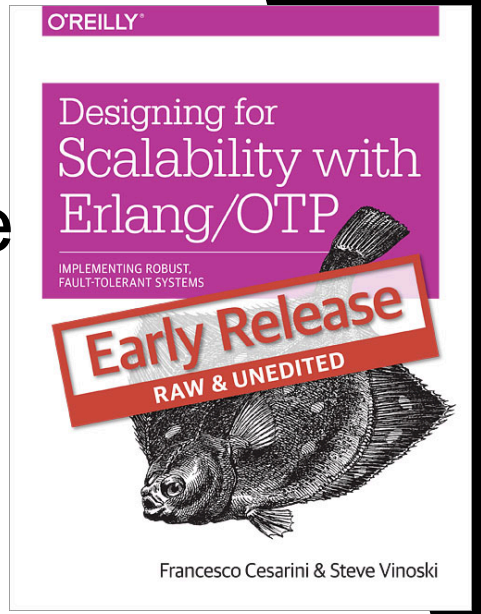
Chapter 13

Ch 13: Node Architecture



Chapter 13

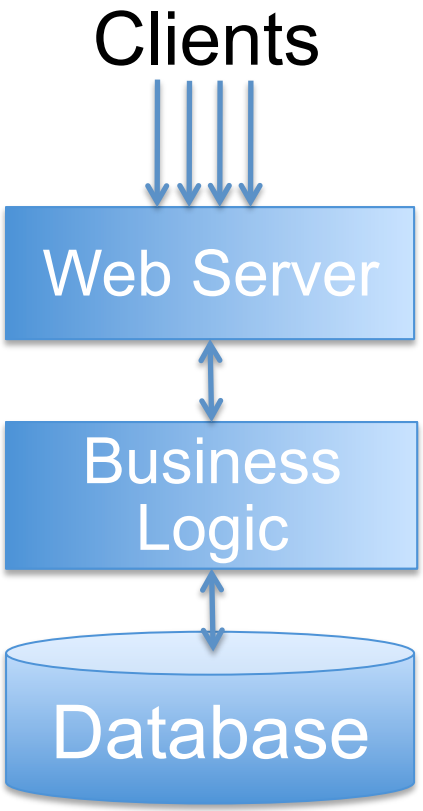
Ch 13: Distributed Architectures
Ch 14: Systems That Never Stop
Ch 15: Scaling Out
Ch 16: Monitoring and Preemptive Support



Distributed Architectures

A node is the smallest executable standalone unit consisting of a running instance of the Erlang runtime system.

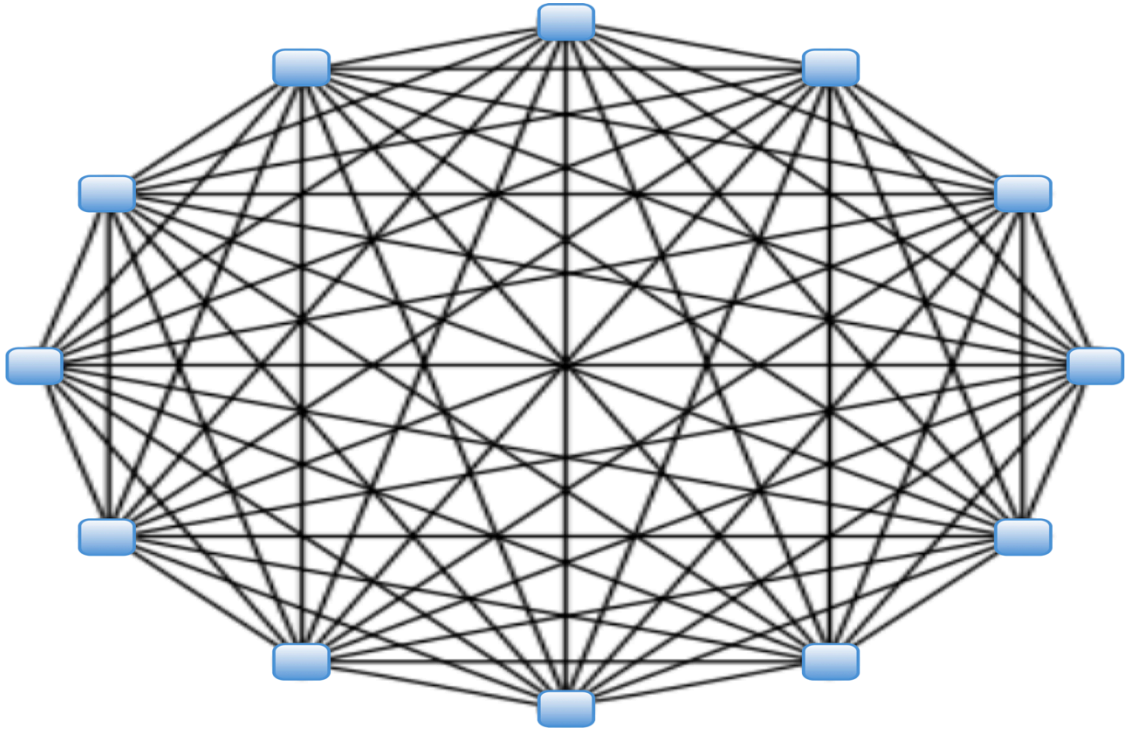
Node Types



Front-end Node

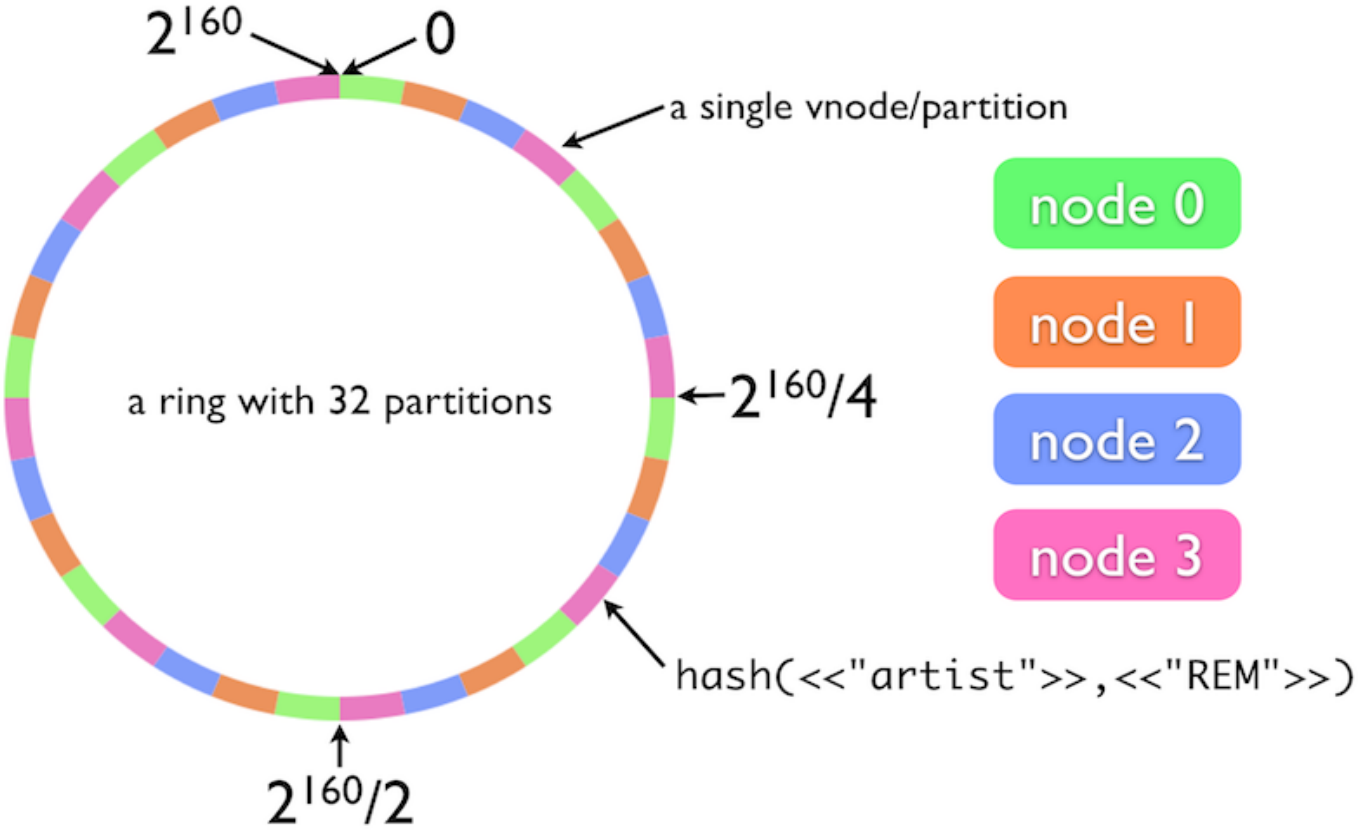
Logic Node

Service Node

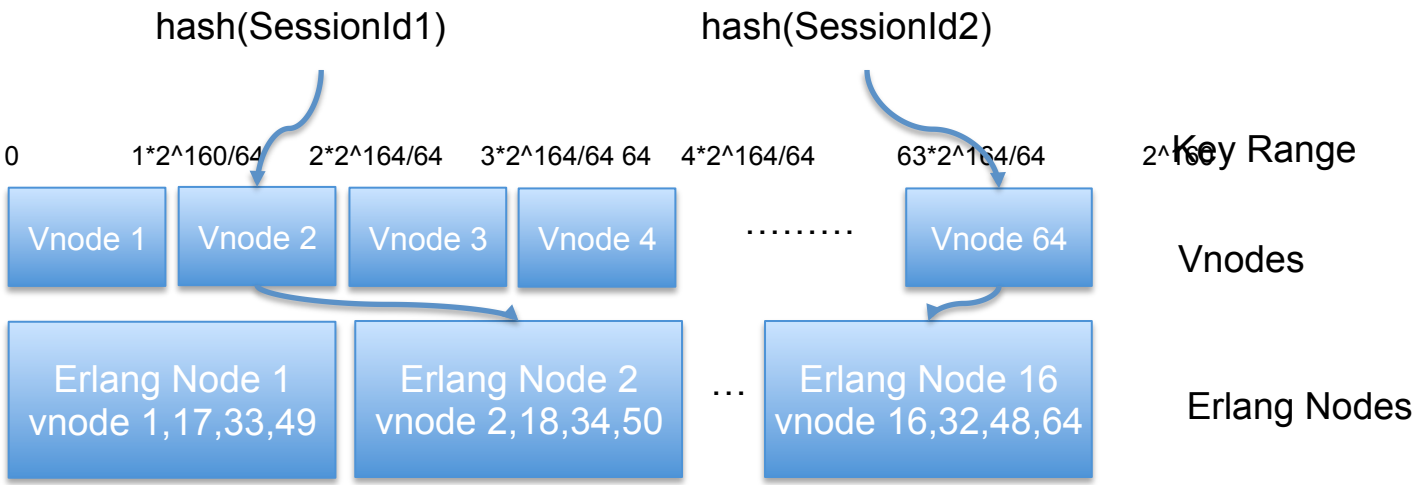


Fully Meshed

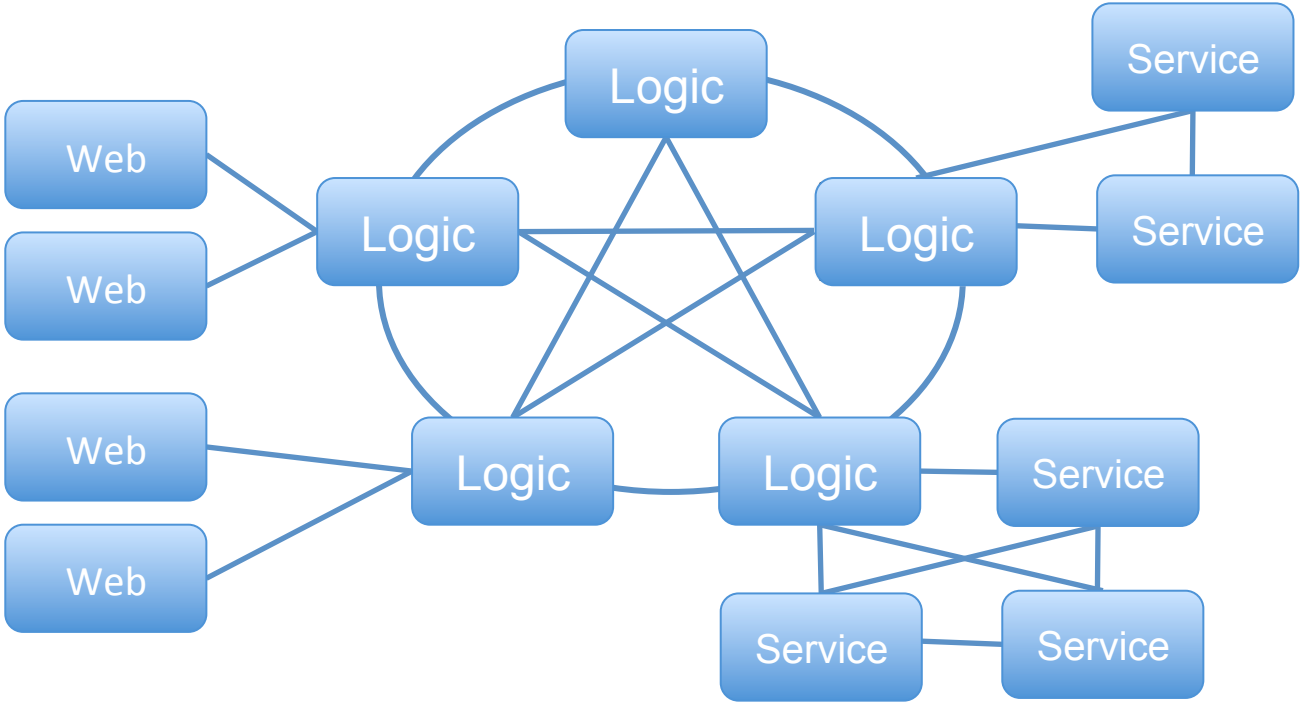
Dynamo


2^160
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
2^160/4
hash(<<"artist">>,<<"REM">>)
2^160/2

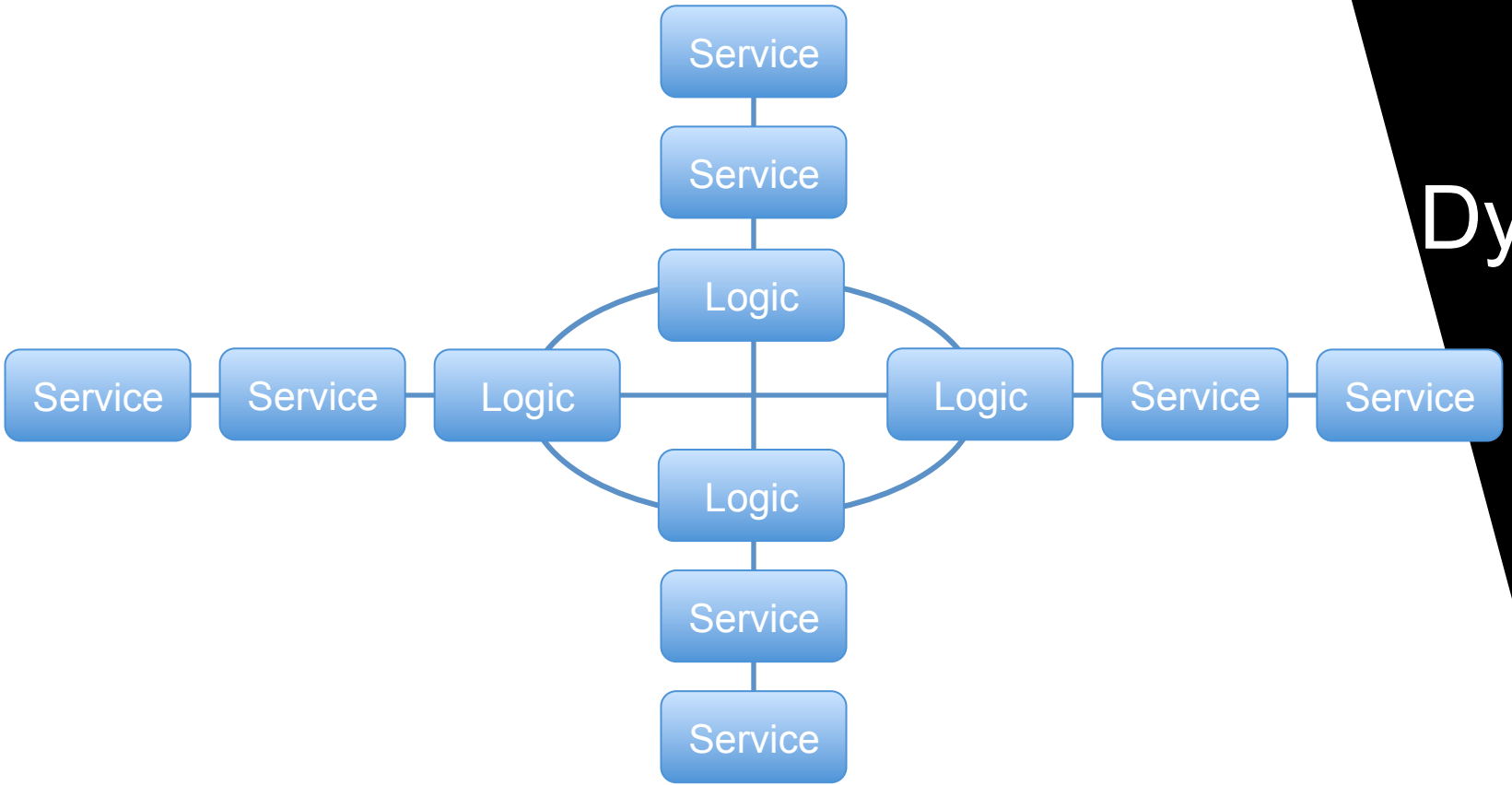
Dynamo


hash(SessionId1)
hash(SessionId2)
0
1*2^160/64
2*2^164/64
3*2^164/64 64
4*2^164/64
63*2^164/64
2^160
Key Range
Vnode 1
Vnode 2
Vnode 3
Vnode 4
.....
Vnode 64
Vnodes
Erlang Node 1
vnode 1,17,33,49
Erlang Node 2
vnode 2,18,34,50
...
Erlang Node 16
vnode 16,32,48,64
Erlang Nodes

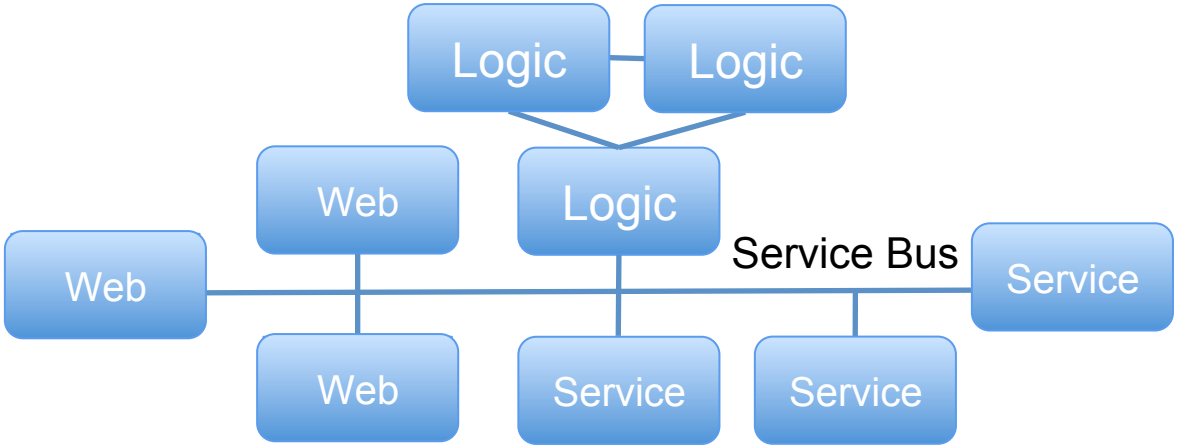
Dynamo


Web
Web
Web
Web
Logic
Logic
Logic
Logic
Logic
Service
Service
Service
Service
Service

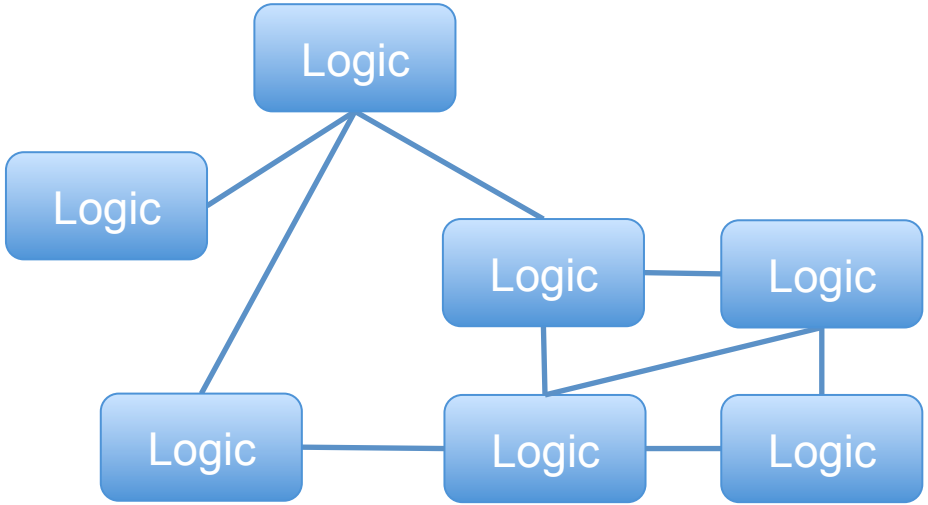
Dynamo


Service
Service
Logic
Service
Service
Logic
Logic
Service
Service
Logic
Service
Service

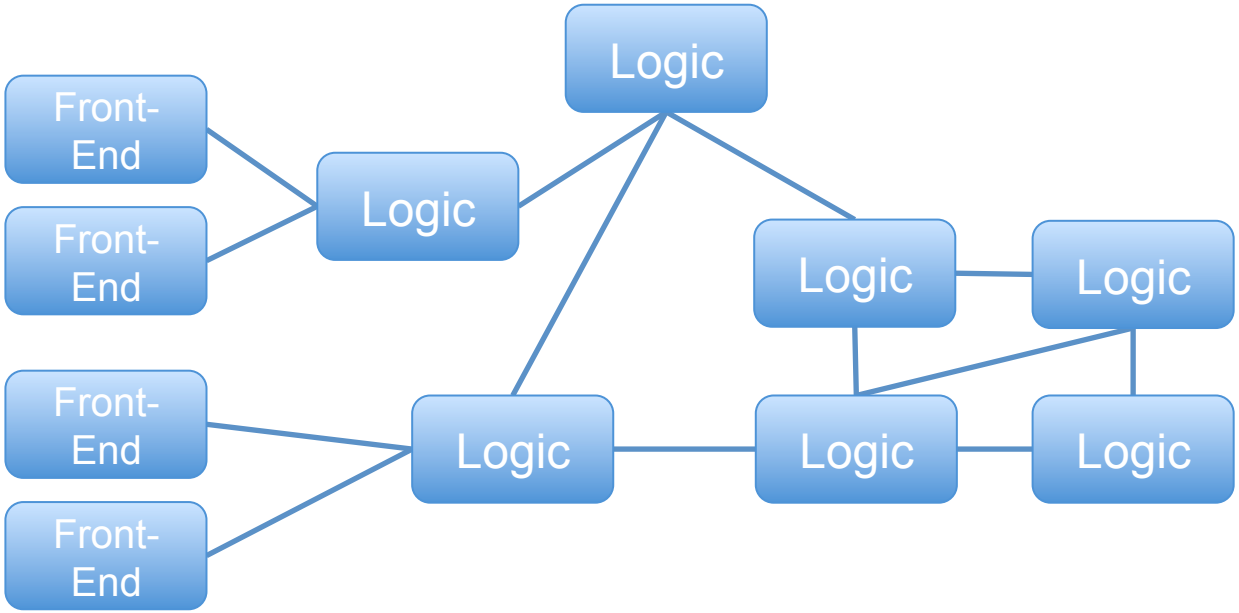
Service Bus


Logic
Logic
Web
Logic
Service Bus
Web
Service
Web
Service
Service

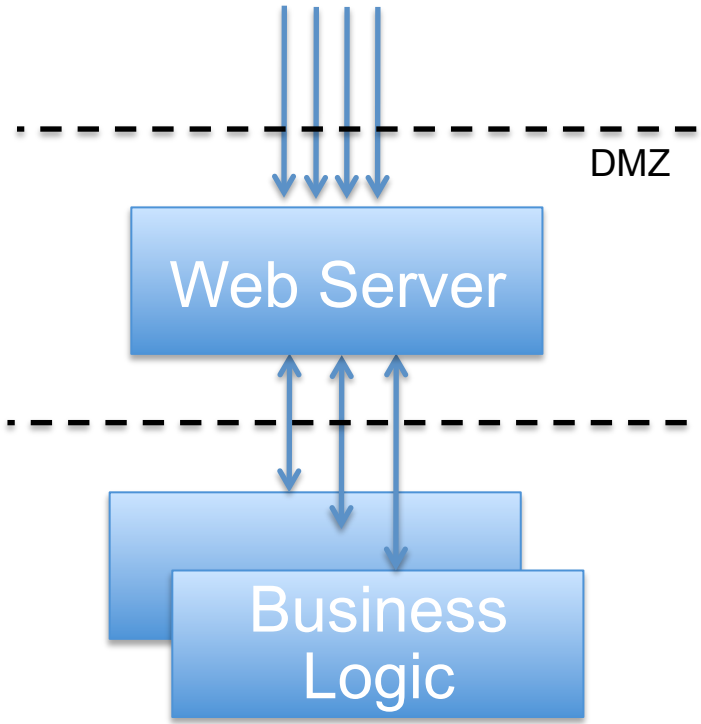
Peer to Peer



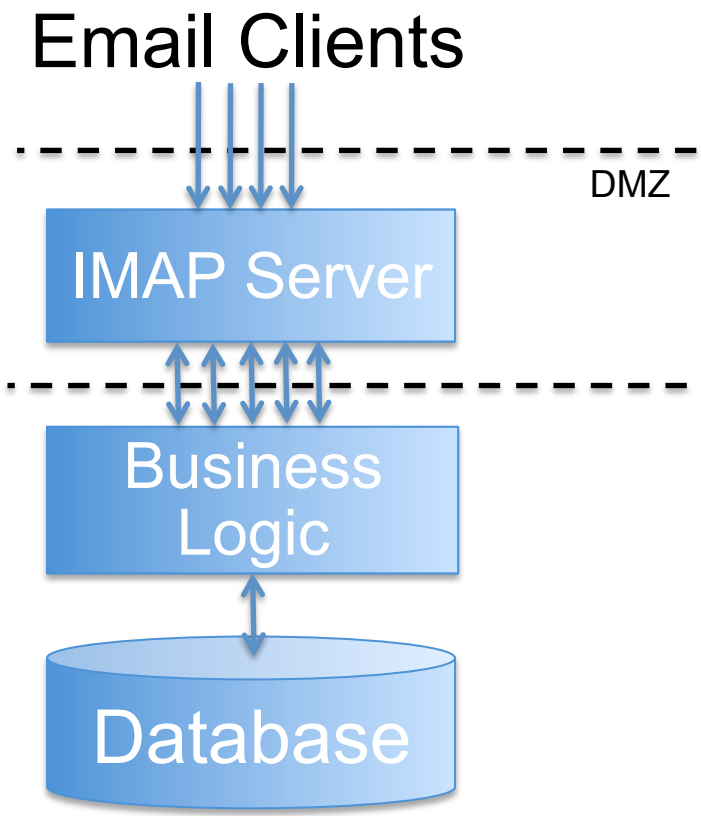
Peer to Peer


Front-End
Front-End
Front-End
Front-End
Logic
Logic
Logic
Logic
Logic
Logic
Logic

Networking

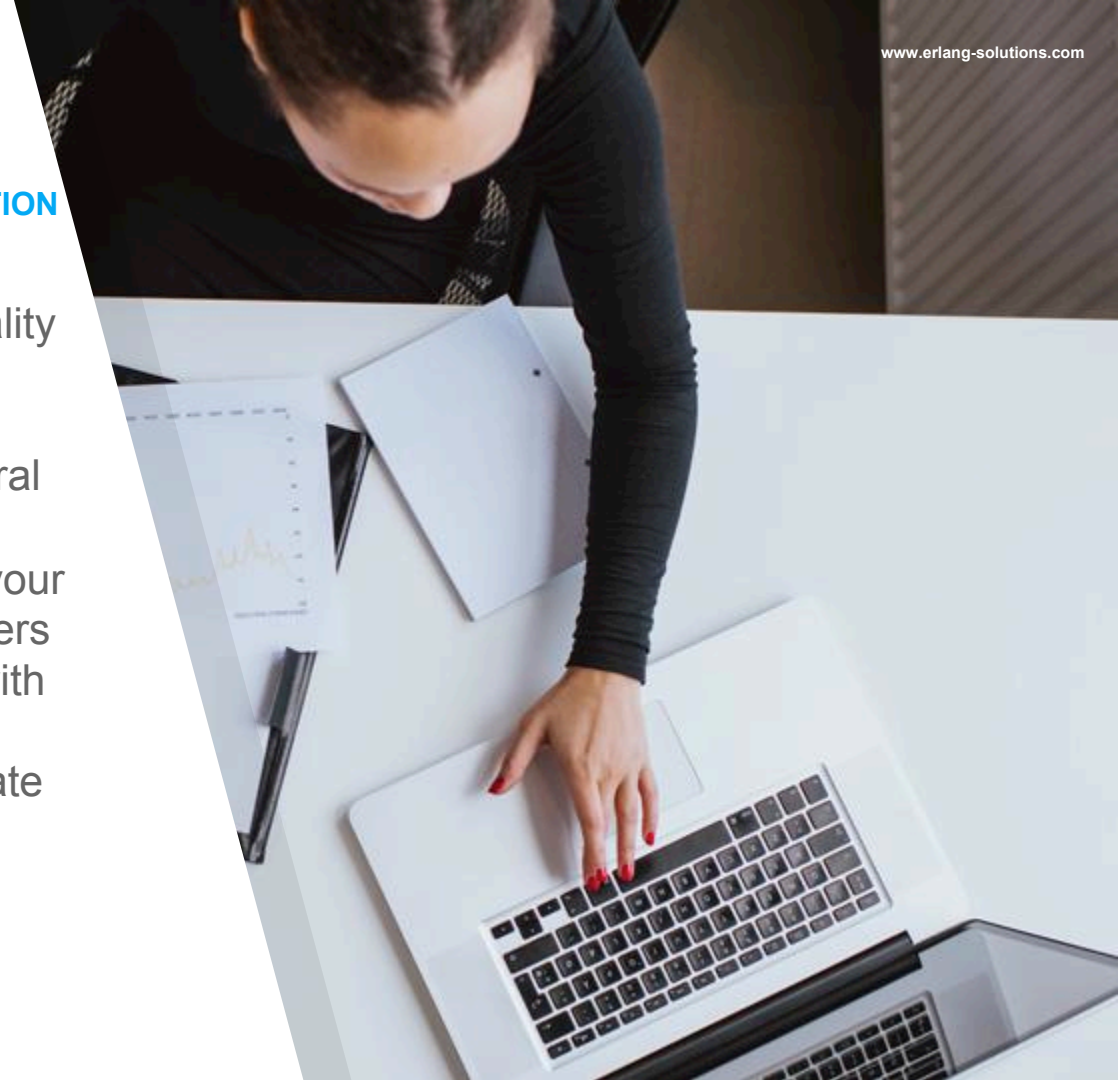


Networking



STEPS EVOLVING AROUND DISTRIBUTION

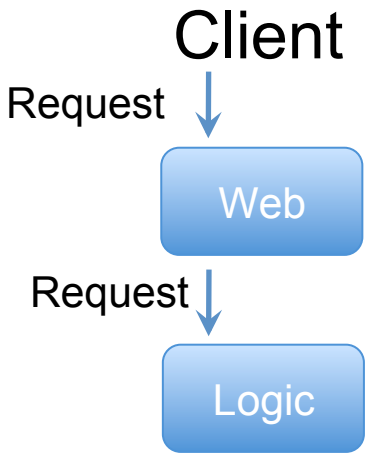
1. Split up your system's functionality into manageable, stand-alone nodes.
2. Choose a distributed architectural pattern.
3. Choose the network protocols your nodes, node families, and clusters will use when communicating with each other.
4. Define your node interfaces, state and data model.

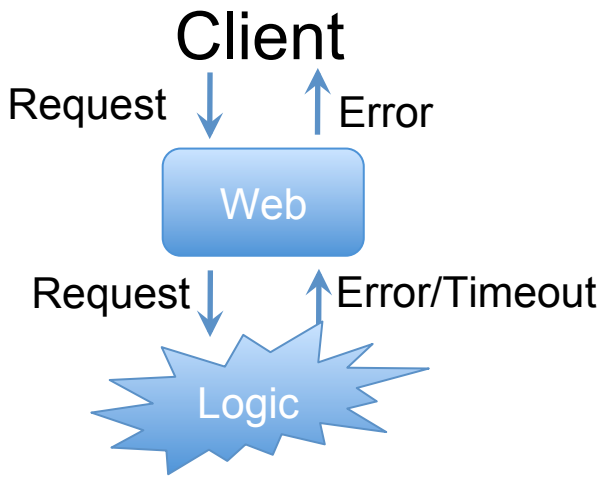


Systems That Never Stop

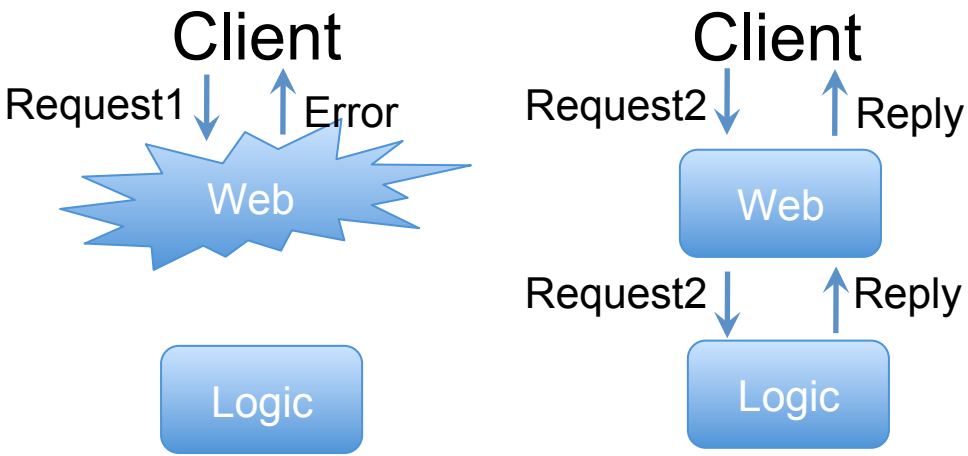
You need at least two computers to make a fault tolerant system.

Fault Tolerance

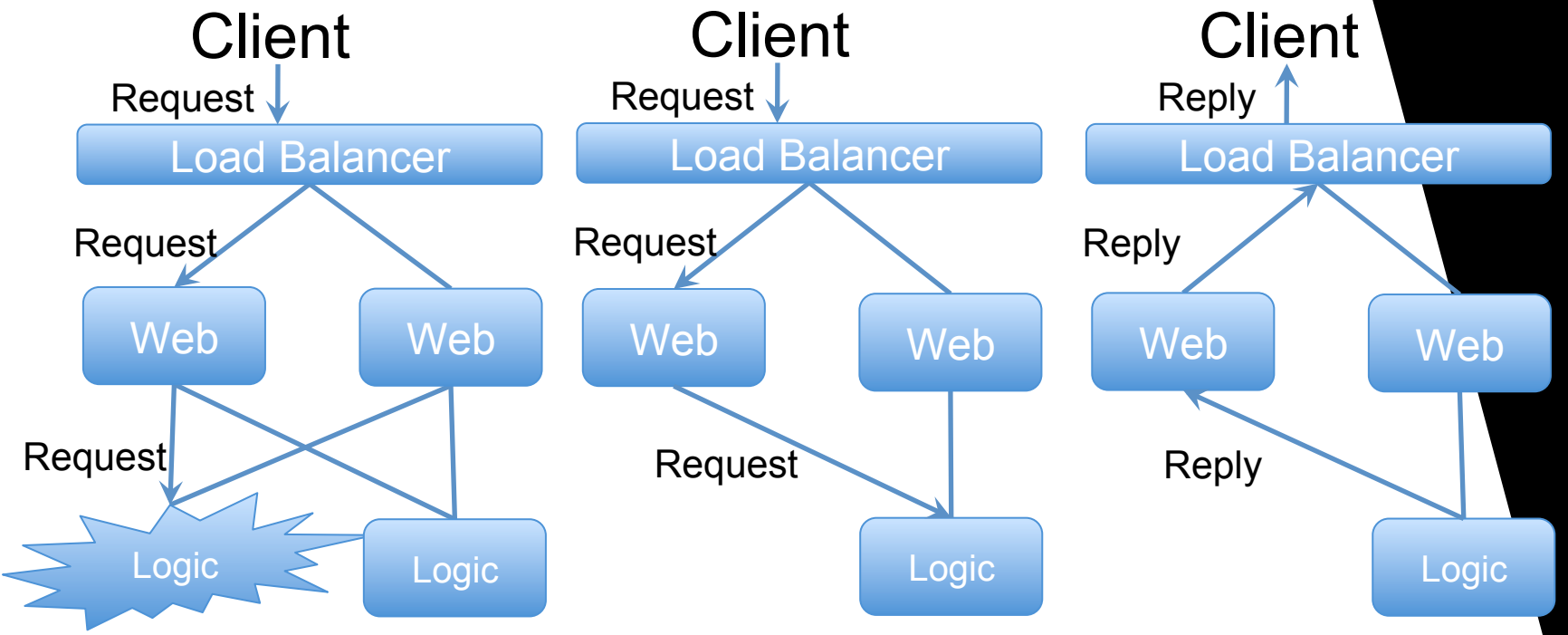




Resilience


Client
Request1
Error
Web
Logic
Client
Request2
Reply
Web
Request2
Reply
Logic

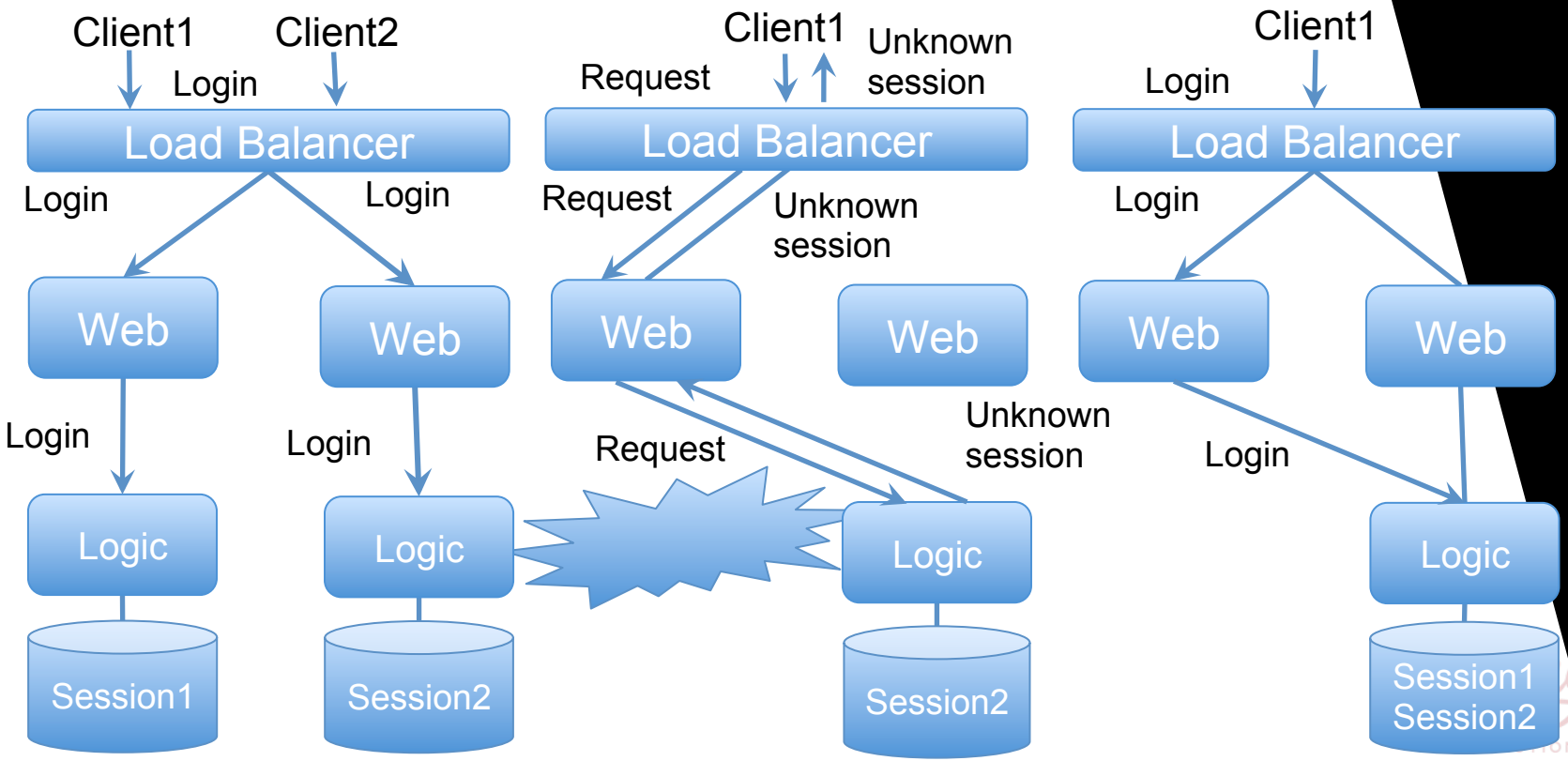
Reliability



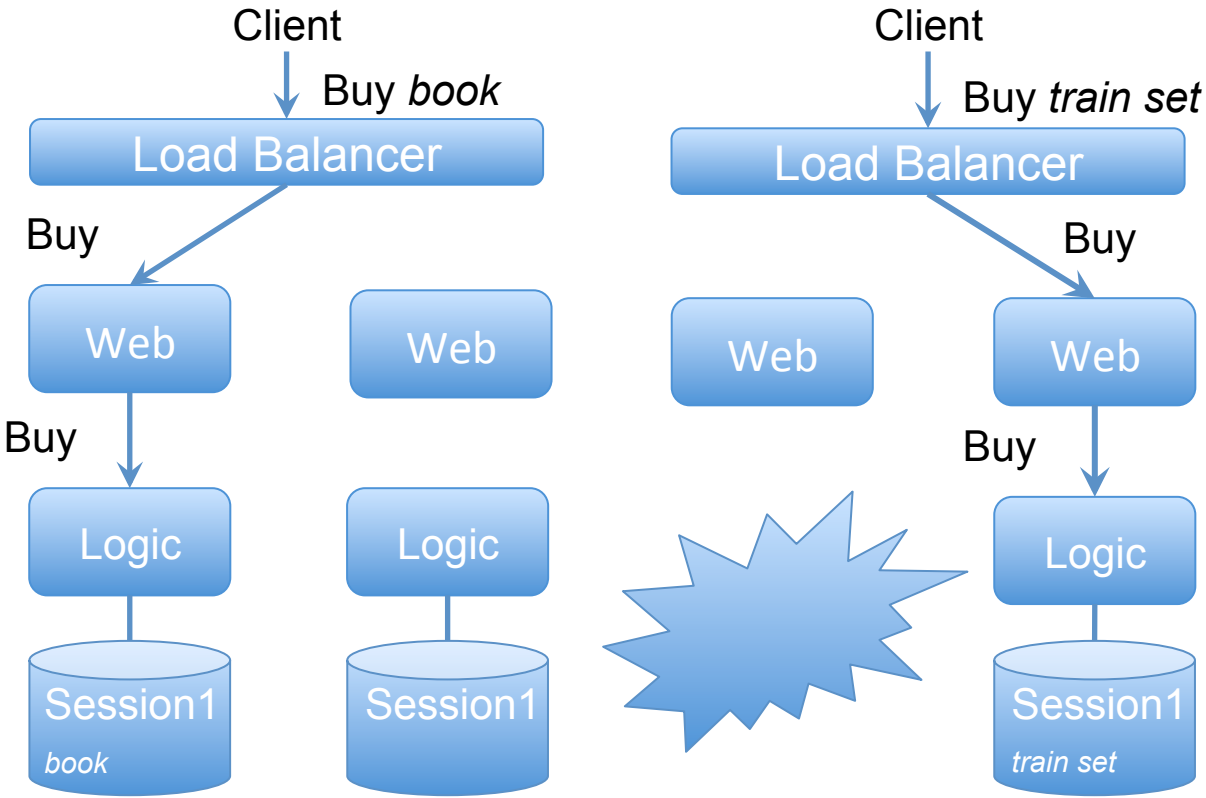
Sharing Data

You have at least two computers to make a fault tolerant system, you need to share state and data.

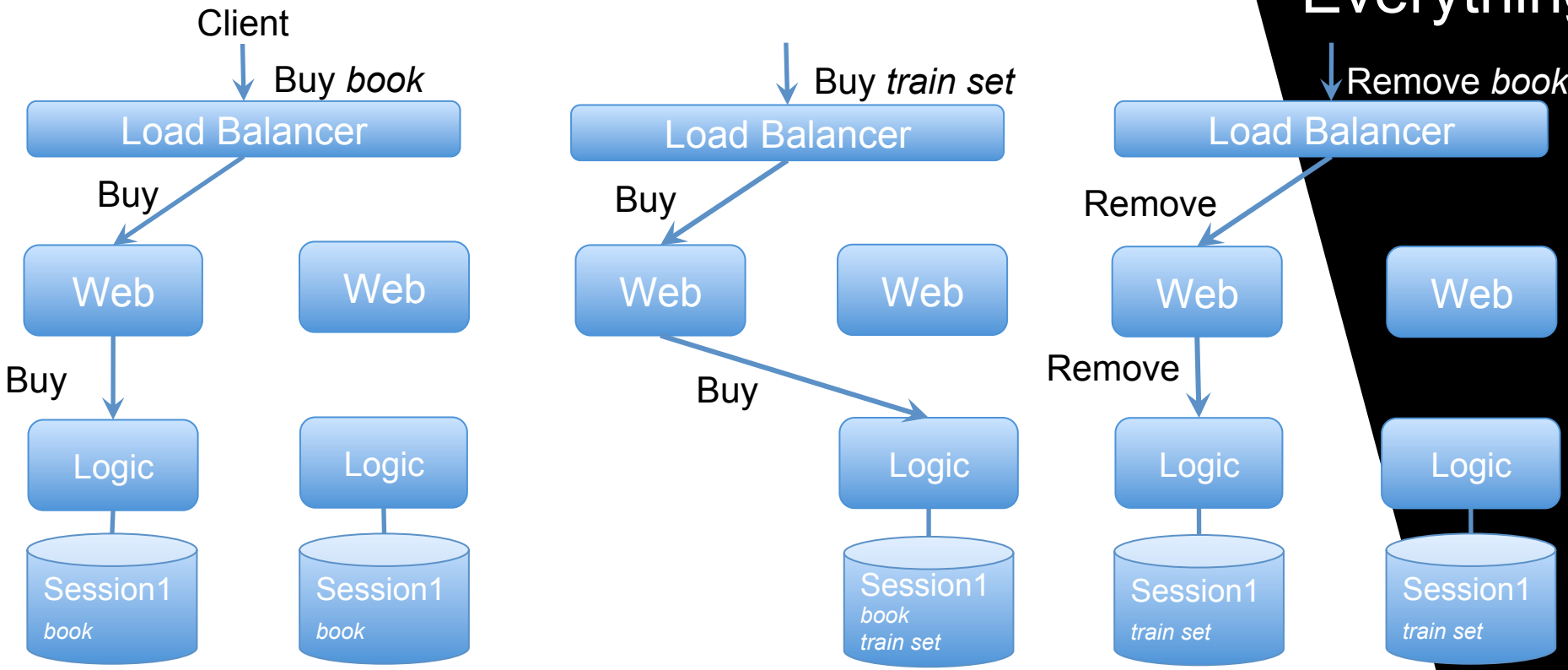
Share Nothing



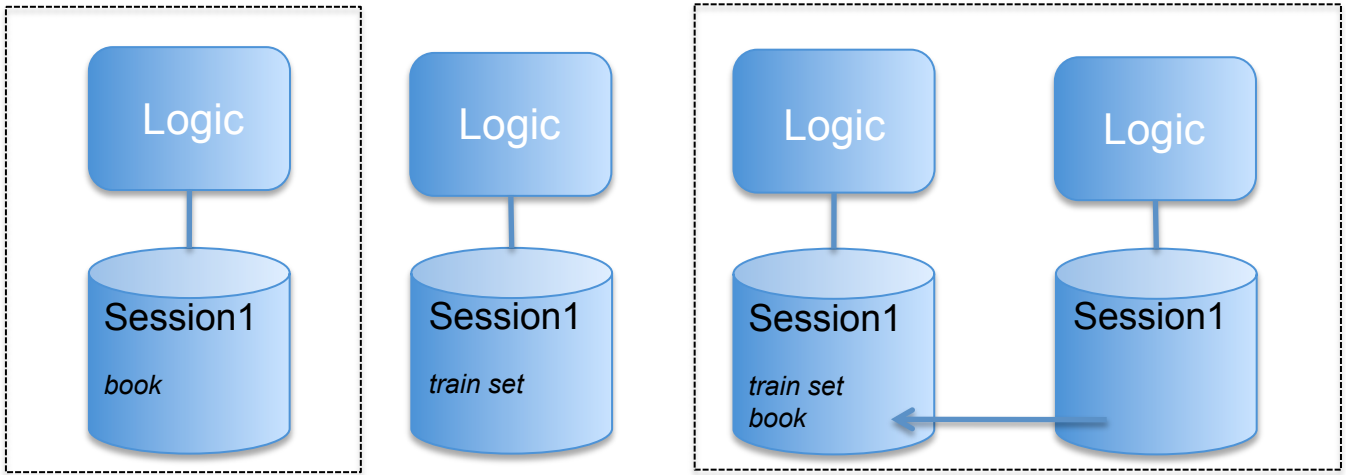
Share Something



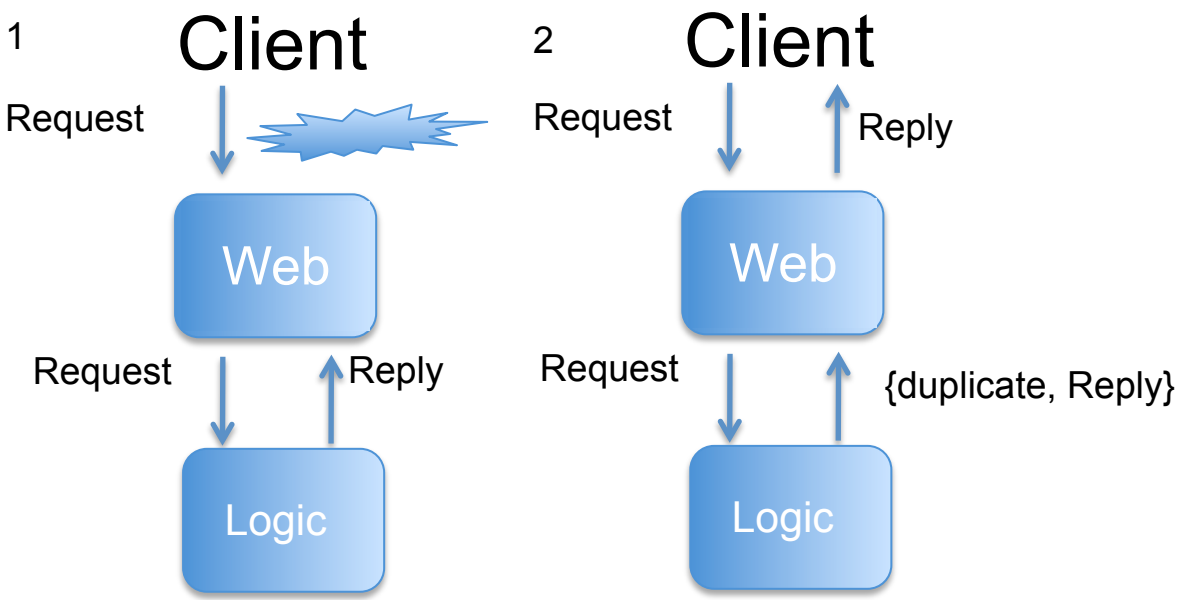
Share Everything



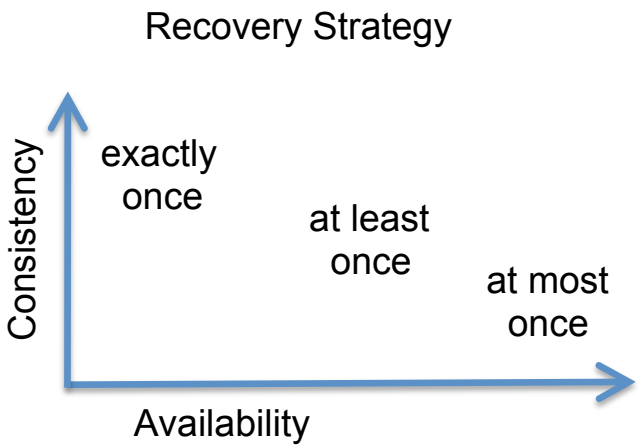
Network Partitions

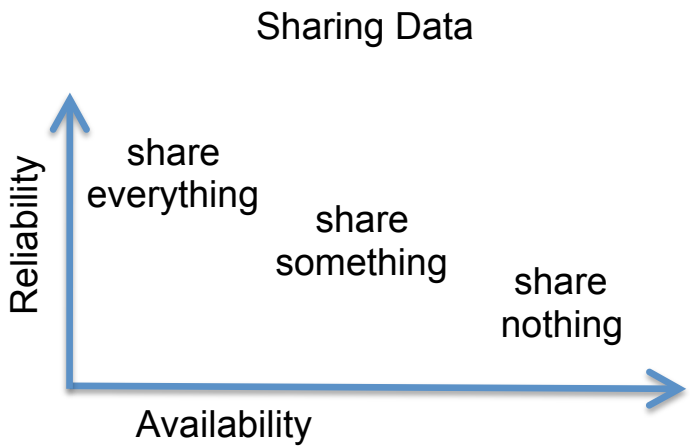


Retry Strategy



Trade-offs


Recovery Strategy
Consistency
exactly once
at least once
at most once
Availability


Sharing Data
Reliability
share everything
share something
share nothing
Availability

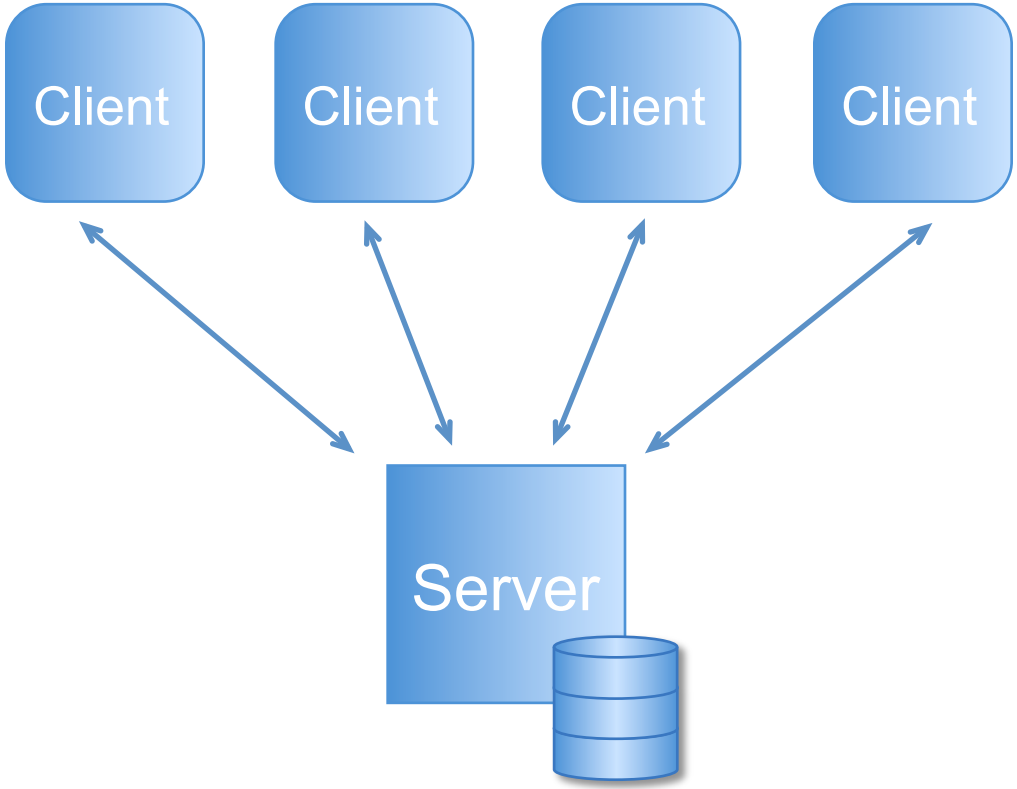
STEPS EVOLVING AROUND AVAILABILITY, CONSISTENCY & RELIABILITY

5. For every interface function in your nodes, you need to pick a retry strategy.
6. For all your data and state, pick your sharing strategy across node families, clusters and types, taking into consideration the needs of your retry strategy.

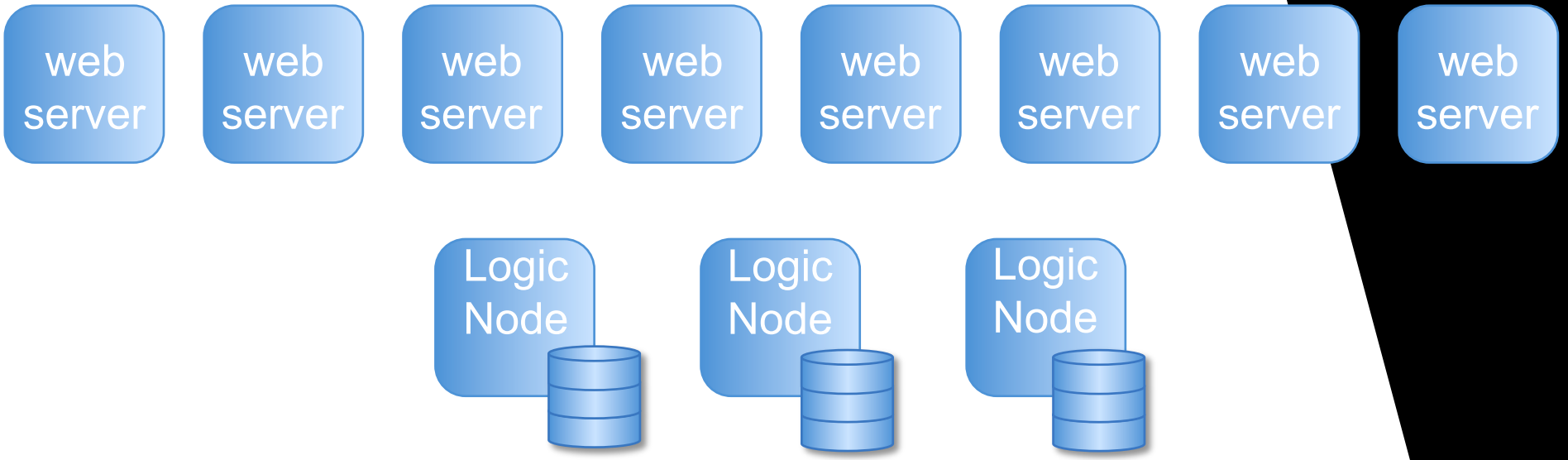
Scaling Out

Distribute for scale and replicate for availability.

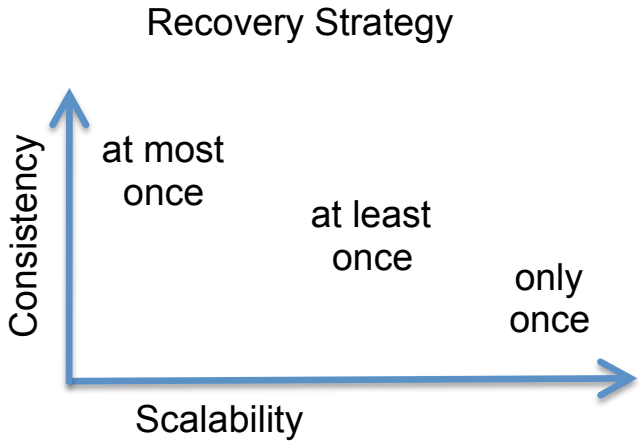
Scaling Vertically

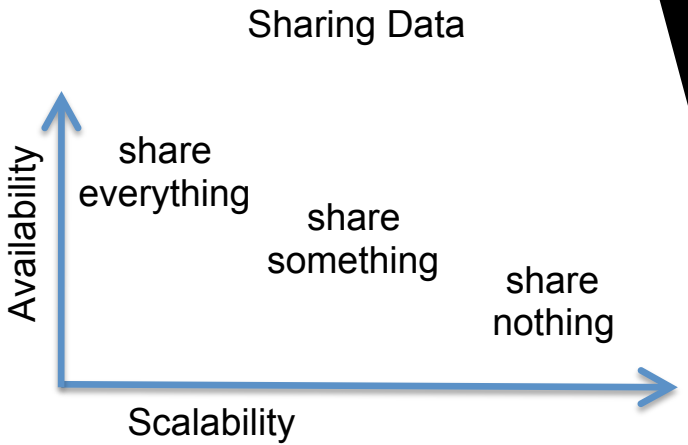


Scaling Horizontally

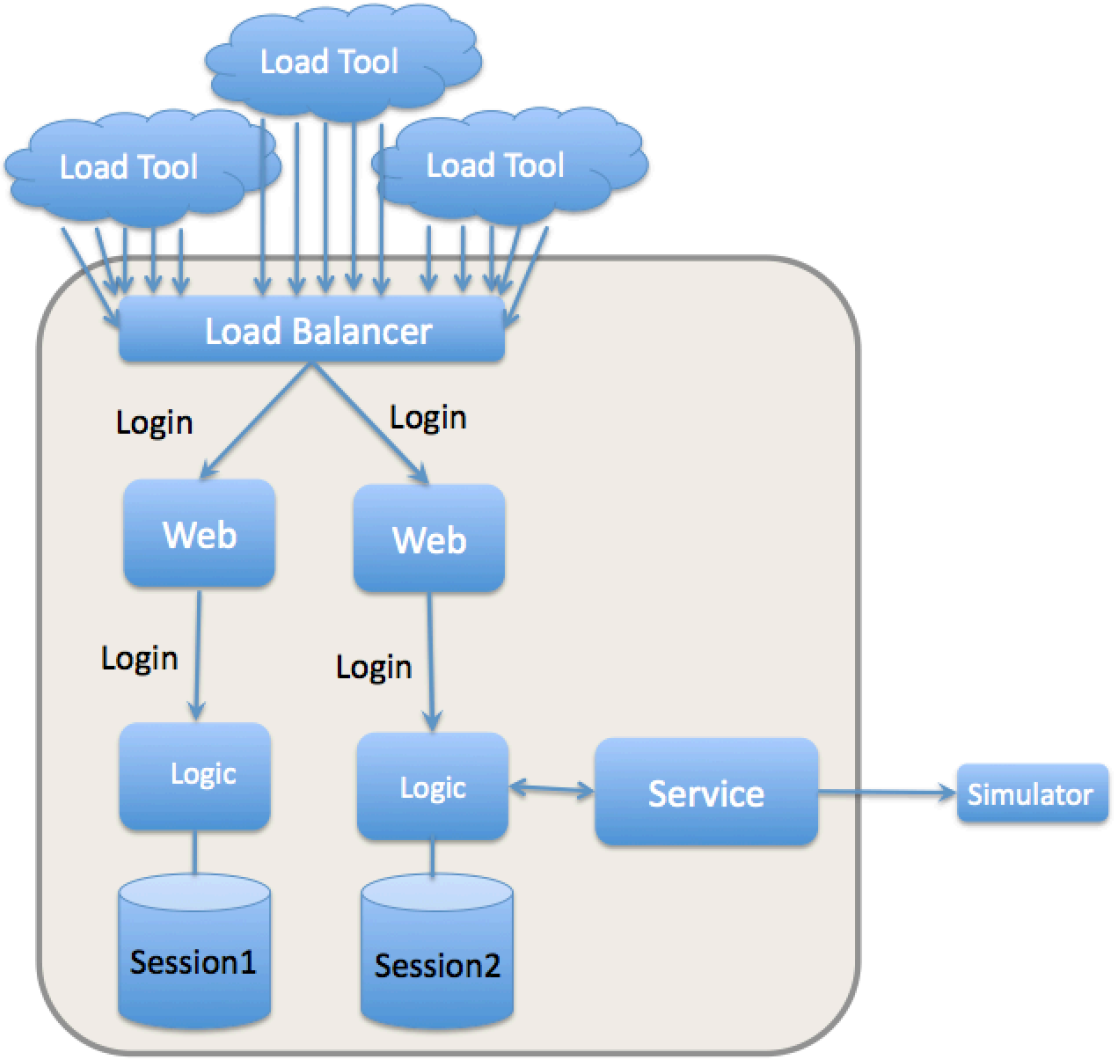


Trade-offs

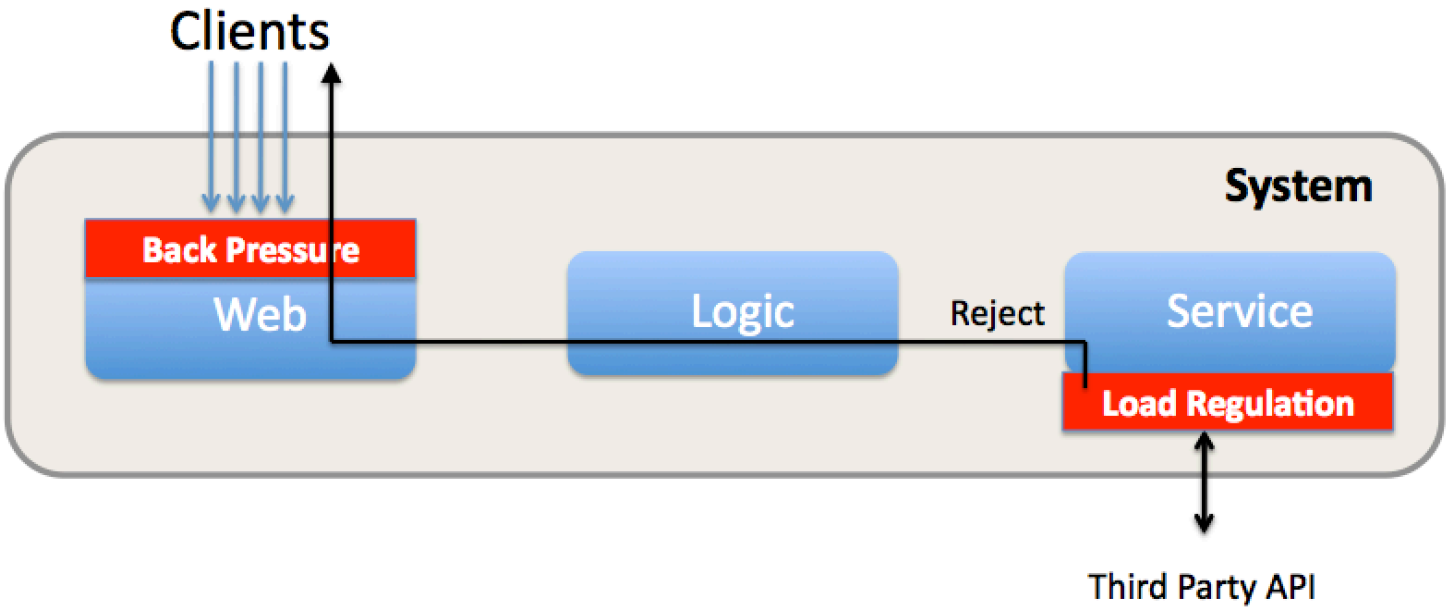

Recovery Strategy
Consistency
at most once
at least once
only once
Scalability


Sharing Data
Availability
share everything
share something
share nothing
Scalability

Capacity Planning



Capacity Planning



- CAPACITY PLANNING -

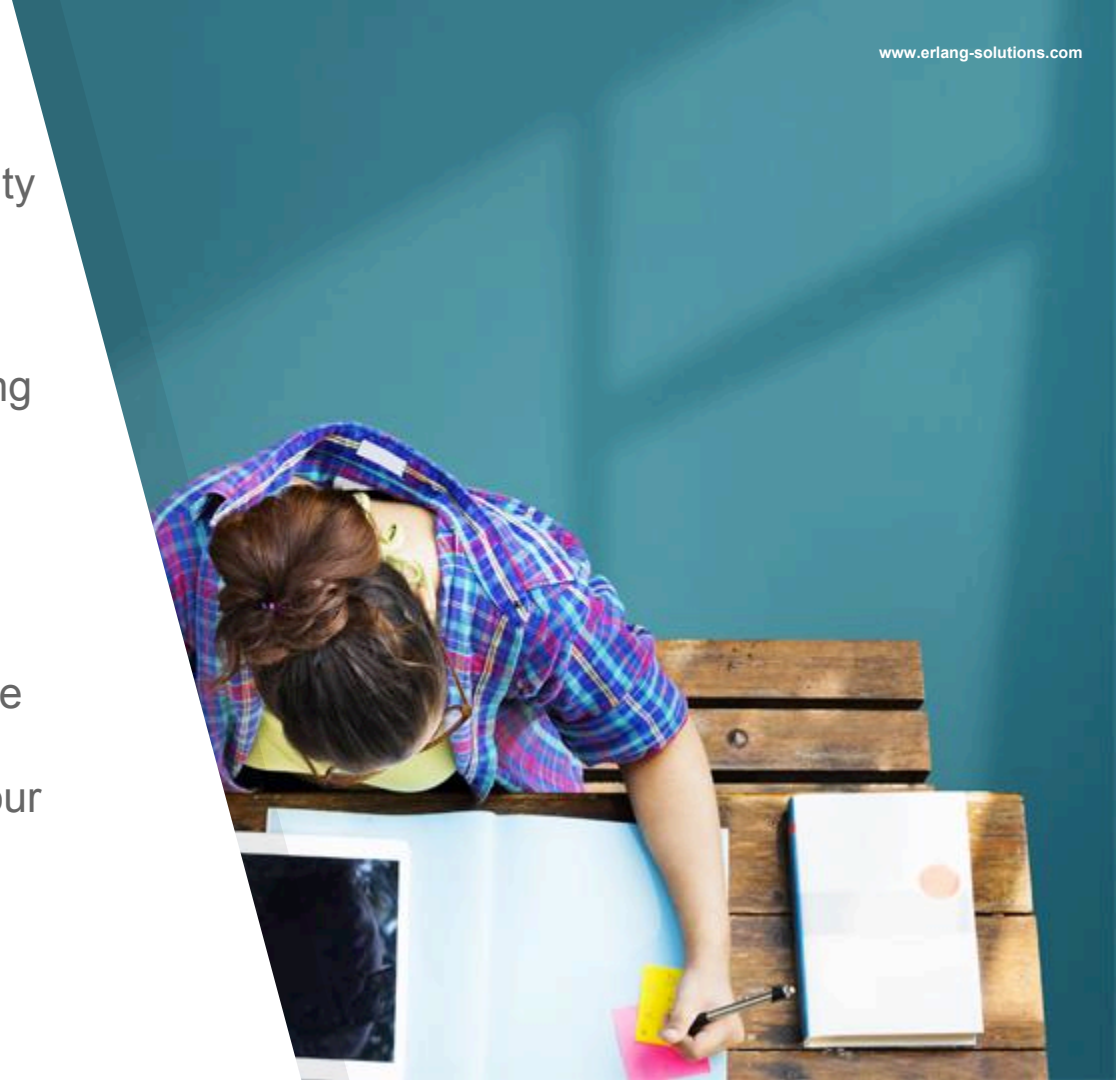
Capacity planning is the design phase which guarantees that your system can withstand the load it was built to handle, and with time, scaling to handle increased demand.

- No single point of failure
- Cluster blueprint for scalability
- Load Regulation
- Back Pressure

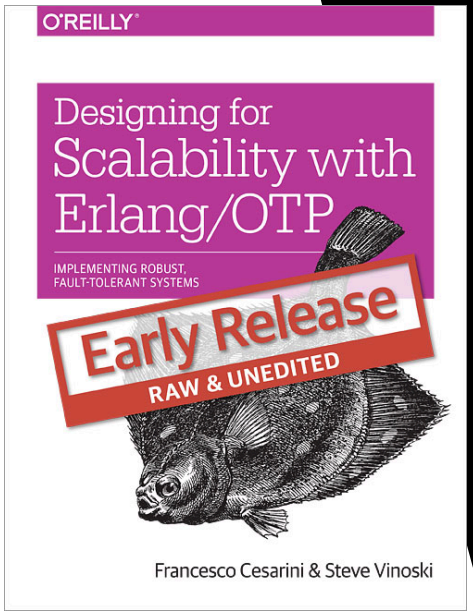
Monitoring and Preemptive Support

With the right tools and approach, the five nines once reserved for Telecom systems are now easily attainable in whatever other vertical for which you might be developing software.

1. Split up your system's functionality into manageable, stand-alone nodes.
2. Decide what distributed architectural pattern you are going to use.
3. Decide what network protocols your nodes, node families and clusters will use when. communicating with each other.
4. Define your node interfaces, state and data model.
5. For every interface function in your nodes, you need to pick a retry strategy.



6. For all your data and state, pick your sharing strategy across node families, clusters and types, taking into consideration the needs of your retry strategy.
7. Reiterate through steps 1, 2, 3, 4, 5 & 6 until you have the trade-offs which suit your specification.
8. Design your cluster blueprint, looking at node ratios for scaling up and down.
9. Indentify where to apply back-pressure and load regulation.
10. Define your O&M approach, defining system and business alarms, logs and metrics.



THANK YOU!

Any questions?

francesco@erlang-solutions.com
www.erlang-solutions.com
@francescoC

Discount Code: **authd**
50% off the Early Release
40% off the printed copy

Erlang SOLUTIONS