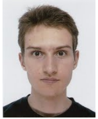
Alex Auvolat, Deuxfleurs Association

`https://garagehq.deuxfleurs.fr/`

Matrix channel: `#garage:deuxfleurs.fr`

Who I am



Alex Auvolat
PhD; co-founder of Deuxfleurs

Deuxfleurs
A non-profit self-hosting collective,
member of the CHATONS network

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting as an alternative to large cloud providers

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting
as an alternative to large cloud providers

Why is it hard?

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting as an alternative to large cloud providers

Why is it hard?

<u>Resilience</u>

(we want good uptime/availability with low supervision)

How to make a stable system

Enterprise-grade systems typically employ:

- RAID
- Redundant power grid + UPS
- Redundant Internet connections
- Low-latency links
- ...

→ it's costly and only worth it at DC scale

How to make a resilient system

Instead, we use:

- Commodity hardware (e.g. old desktop PCs)

How to make a resilient system

How to make a resilient system

How to make a resilient system

Instead, we use:

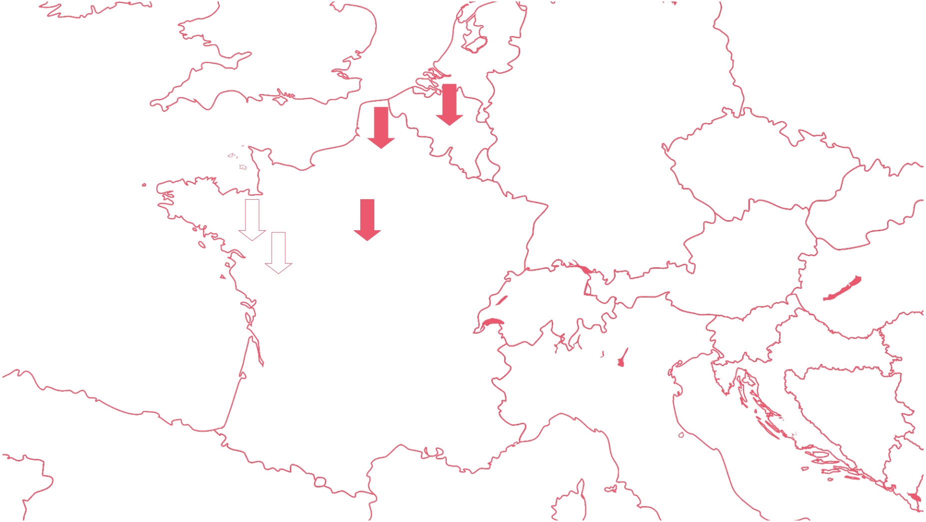
- Commodity hardware (e.g. old desktop PCs)
- Commodity Internet (e.g. FTTB, FTTH) and power grid

How to make a resilient system

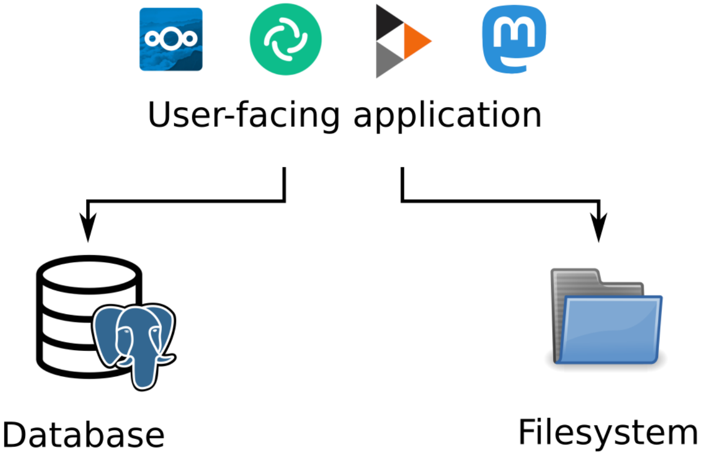
Instead, we use:

- Commodity hardware (e.g. old desktop PCs)
- Commodity Internet (e.g. FTTB, FTTH) and power grid
- **Geographical redundancy** (multi-site replication)

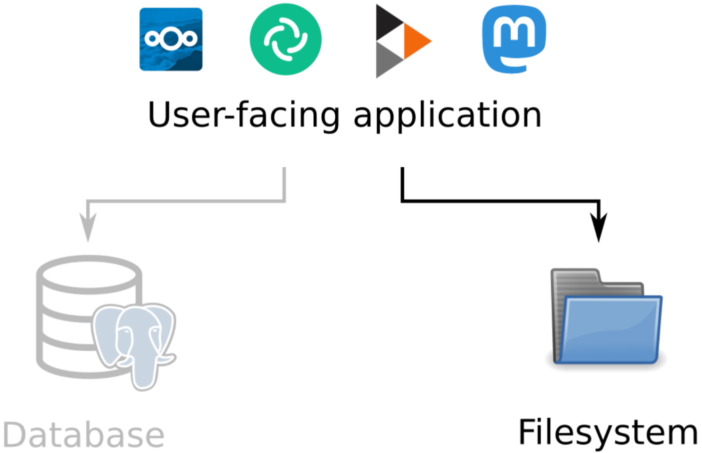
How to make a resilient system



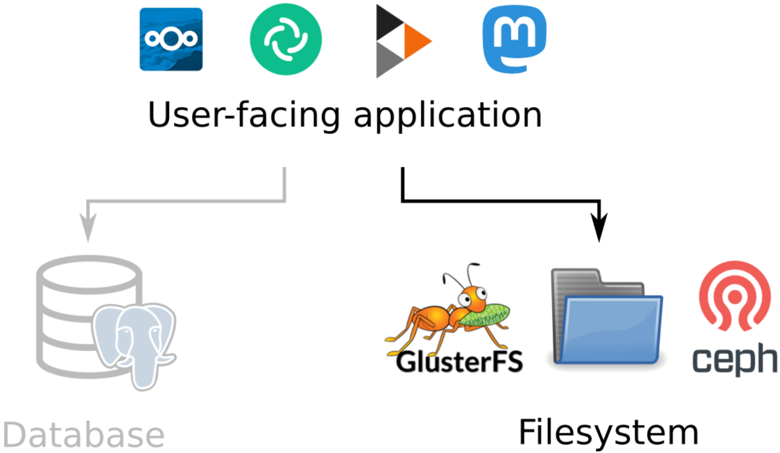
How to make this happen



How to make this happen



How to make this happen



Distributed file systems are slow

File systems are complex, for example:

- Concurrent modification by several processes
- Folder hierarchies
- Other requirements of the POSIX spec (e.g. locks)

Coordination in a distributed system is costly

Costs explode with commodity hardware / Internet connections
(we experienced this!)

A simpler solution: object storage

Only two operations:

- Put an object at a key
- Retrieve an object from its key

(and a few others)

Sufficient for many applications!

A simpler solution: object storage

S3: a de-facto standard, many compatible applications

MinIO is self-hostable but not suited for geo-distributed deployments

Garage is a self-hosted drop-in replacement for the Amazon S3 object store

The data model of object storage

Object storage is basically a key-value store:

Key: file path + name	**Value: file data + metadata**

`index.html`	`Content-Type: text/html; charset=utf-8`
	`Content-Length: 24929`
	`<binary blob>`

`img/logo.svg`	`Content-Type: text/svg+xml`
	`Content-Length: 13429`
	`<binary blob>`

`download/index.html`	`Content-Type: text/html; charset=utf-8`
	`Content-Length: 26563`
	`<binary blob>`

Two big problems

1. **How to place data on different nodes?**

   Constraints: heterogeneous hardware
   Objective: n copies of everything, maximize usable capacity, maximize resilience

   → the Dynamo model + optimization algorithms

Two big problems

1. **How to place data on different nodes?**

   Constraints: heterogeneous hardware
   Objective: n copies of everything, maximize usable capacity, maximize resilience

   → the Dynamo model + optimization algorithms

2. **How to guarantee consistency?**

   Constraints: slow network (geographical distance), node unavailability/crashes
   Objective: maximize availability, read-after-write guarantee

   → CRDTs, monotonicity, read and write quorums

Problem 1: placing data

Key-value stores, upgraded: the Dynamo model

Two keys:

- Partition key: used to divide data into partitions (a.k.a. shards)
- Sort key: used to identify items inside a partition

Partition key: bucket	Sort key: filename	Value
`website`	`index.html`	(file data)
`website`	`img/logo.svg`	(file data)
`website`	`download/index.html`	(file data)
`backup`	`borg/index.2822`	(file data)
`backup`	`borg/data/2/2329`	(file data)
`backup`	`borg/data/2/2680`	(file data)
`private`	`qq3a2nbe1qjq0ebbvo6ocsp6co`	(file data)

Key-value stores, upgraded: the Dynamo model

- Data with different partition keys is stored independently, on a different set of nodes

  → no easy way to list all partition keys
  → no cross-shard transactions

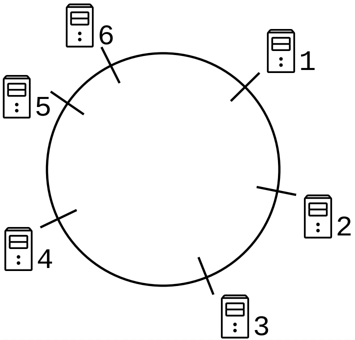
- Placing data: hash the partition key, select nodes accordingly

  → distributed hash table (DHT)

- For a given value of the partition key, items can be listed using their sort keys

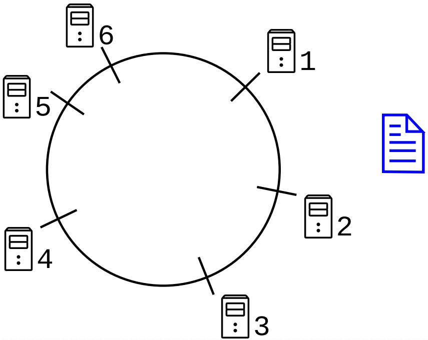
How to spread files over different cluster nodes?

Consistent hashing (Dynamo):



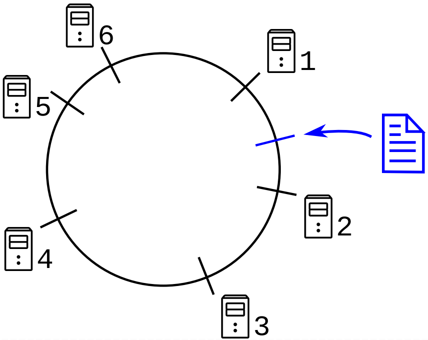
How to spread files over different cluster nodes?

Consistent hashing (Dynamo):



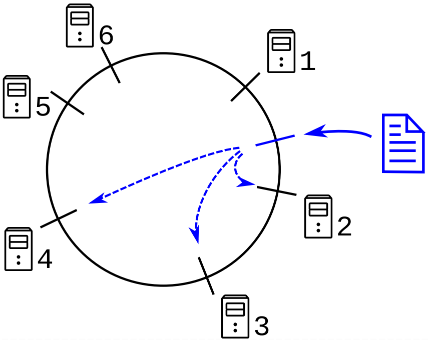
How to spread files over different cluster nodes?

Consistent hashing (Dynamo):



How to spread files over different cluster nodes?

Consistent hashing (Dynamo):

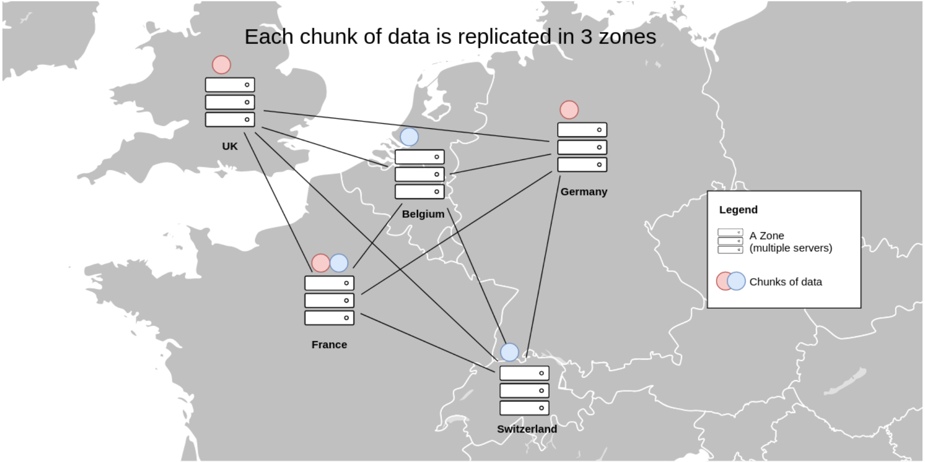


Constraint: location-awareness

```
alex@io:~$ docker exec -ti garage /garage status
==== HEALTHY NODES ====
ID                Hostname   Address                       Tags                  Zone     Capacity
7d50f042280fea98  io         [2a01:e0a:5e4:1d0::57]:3901   [io,jupiter]          jupiter  20
d9b5959e58a3ab8c  drosera    [2a01:e0a:260:b5b0::4]:3901   [drosera,atuin]       atuin    20
966dfc7ed8049744  datura     [2a01:e0a:260:b5b0::2]:3901   [datura,atuin]        atuin    10
8cf284e7df17d0fd  celeri     [2a06:a004:3025:1::33]:3901   [celeri,neptune]      neptune  5
156d0f7a88b1e328  digitale   [2a01:e0a:260:b5b0::3]:3901   [digitale,atuin]      atuin    10
5fcb3b6e39db3dcb  concombre  [2a06:a004:3025:1::31]:3901   [concombre,neptune]   neptune  5
a717e5b618267806  courgette  [2a06:a004:3025:1::32]:3901   [courgette,neptune]   neptune  5
alex@io:~$
```

Garage replicates data on different zones when possible

Constraint: location-awareness



Issues with consistent hashing

- Consistent hashing doesn't dispatch data based on geographical location of nodes

Issues with consistent hashing

- Consistent hashing doesn't dispatch data based on geographical location of nodes
- Geographically aware adaptation, try 1:
  data quantities not well balanced between nodes

Issues with consistent hashing

- Consistent hashing doesn't dispatch data based on geographical location of nodes
- Geographically aware adaptation, try 1:
  data quantities not well balanced between nodes
- Geographically aware adaptation, try 2:
  too many reshuffles when adding/removing nodes

How to spread files over different cluster nodes?

Garage's method: build an index table

Realization: we can actually precompute an optimal solution

How to spread files over different cluster nodes?

Garage's method: build an index table

Realization: we can actually precompute an optimal solution

Partition	**Node 1**	**Node 2**	**Node 3**
Partition 0	Io (jupiter)	Drosera (atuin)	Courgette (neptune)
Partition 1	Datura (atuin)	Courgette (neptune)	Io (jupiter)
Partition 2	Io(jupiter)	Celeri (neptune)	Drosera (atuin)
⋮	⋮	⋮	⋮
Partition 255	Concombre (neptune)	Io (jupiter)	Drosera (atuin)

How to spread files over different cluster nodes?

Garage's method: build an index table

Realization: we can actually precompute an optimal solution

Partition	**Node 1**	**Node 2**	**Node 3**
Partition 0	Io (jupiter)	Drosera (atuin)	Courgette (neptune)
Partition 1	Datura (atuin)	Courgette (neptune)	Io (jupiter)
Partition 2	Io(jupiter)	Celeri (neptune)	Drosera (atuin)
⋮	⋮	⋮	⋮
Partition 255	Concombre (neptune)	Io (jupiter)	Drosera (atuin)

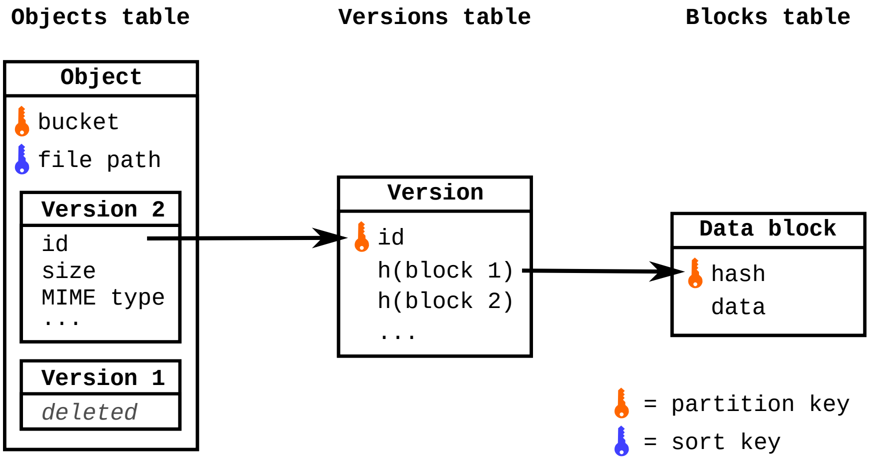
The index table is built centrally using an optimal algorithm,
then propagated to all nodes

The relationship between *partition* and *partition key*

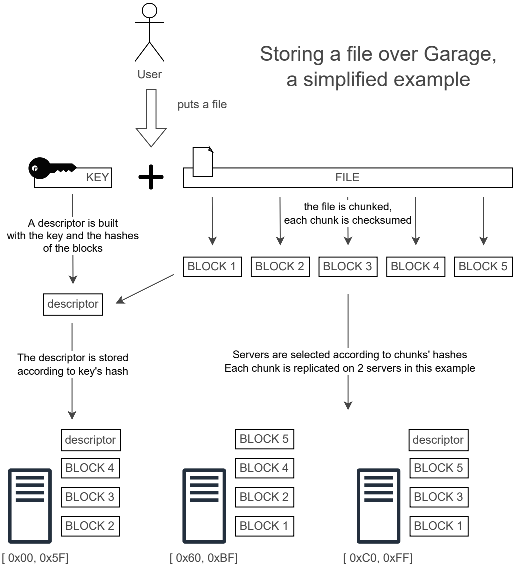
Partition key	Partition	Sort key	Value
`website`	Partition 12	`index.html`	(file data)
`website`	Partition 12	`img/logo.svg`	(file data)
`website`	Partition 12	`download/index.html`	(file data)
`backup`	Partition 42	`borg/index.2822`	(file data)
`backup`	Partition 42	`borg/data/2/2329`	(file data)
`backup`	Partition 42	`borg/data/2/2680`	(file data)
`private`	Partition 42	`qq3a2nbe1qjq0ebbvo6ocsp6co`	(file data)

To read or write an item: hash partition key
$\rightarrow$ determine partition number (first 8 bits)
$\rightarrow$ find associated nodes

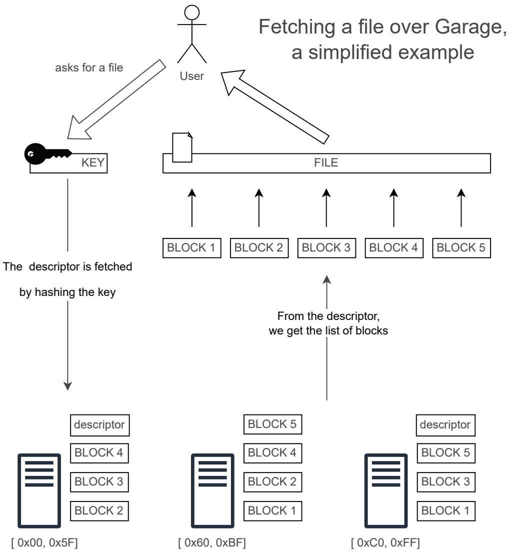
Garage's internal data structures



Storing and retrieving files



Storing and retrieving files



Problem 2: ensuring consistency

Consensus vs weak consistency

Consensus-based systems:

- **Leader-based:** a leader is elected to coordinate all reads and writes
- **Linearizability** of all operations (strongest consistency guarantee)
- Any sequential specification can be implemented as a **replicated state machine**
- **Costly**, the leader is a bottleneck; leader elections on failure take time

Consensus vs weak consistency

Consensus-based systems:

- **Leader-based:** a leader is elected to coordinate all reads and writes
- **Linearizability** of all operations (strongest consistency guarantee)
- Any sequential specification can be implemented as a **replicated state machine**
- **Costly**, the leader is a bottleneck; leader elections on failure take time

Weakly consistent systems:

- **Nodes are equivalent**, any node can originate a read or write operation
- **Read-after-write consistency** with quorums, eventual consistency without
- **Operations have to commute**, i.e. we can only implement CRDTs
- **Fast**, no single bottleneck; works the same with offline nodes

Consensus vs weak consistency

From a theoretical point of view:

Consensus-based systems:

Require **additional assumptions** such as a fault detector or a strong RNG (FLP impossibility theorem)

Weakly consistent systems:

Can be implemented in **any asynchronous message passing distributed system** with node crashes

They represent **different classes of computational capability**

Consensus vs weak consistency

The same objects cannot be implemented in both models.

Consensus-based systems:

Any sequential specification

Easier to program for: just write your program as if it were sequential on a single machine

Weakly consistent systems:

Only CRDTs
(conflict-free replicated data types)

Part of the complexity is **reported to the consumer of the API**

Understanding the power of consensus

Consensus: an API with a single operation, $\mathit{propose}(x)$

1. nodes all call $\mathit{propose}(x)$ with their proposed value;
2. nodes all receive the same value as a return value, which is one of the proposed values

Understanding the power of consensus

Consensus: an API with a single operation, *propose*(x)

1. nodes all call *propose*(x) with their proposed value;
2. nodes all receive the same value as a return value, which is one of the proposed values

Equivalent to a distributed algorithm that gives a total order on all requests

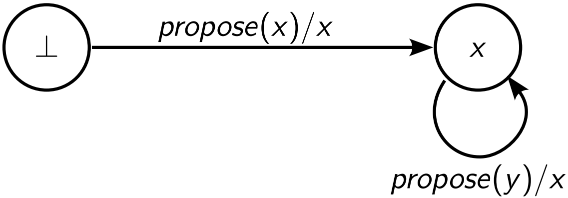
Understanding the power of consensus

Consensus: an API with a single operation, *propose*(x)

1. nodes all call *propose*(x) with their proposed value;
2. nodes all receive the same value as a return value, which is one of the proposed values

Equivalent to a distributed algorithm that gives a total order on all requests

Implemented by this simple replicated state machine:



Can my object be implemented without consensus?

Given the specification of an API:

- **Using this API, we can implement the consensus object** (the *propose* function)
  $\rightarrow$ the API is equivalent to consensus/total ordering of messages
  $\rightarrow$ the API cannot be implemented in a weakly consistent system

Can my object be implemented without consensus?

Given the specification of an API:

- **Using this API, we can implement the consensus object** (the *propose* function)
  $\rightarrow$ the API is equivalent to consensus/total ordering of messages
  $\rightarrow$ the API cannot be implemented in a weakly consistent system

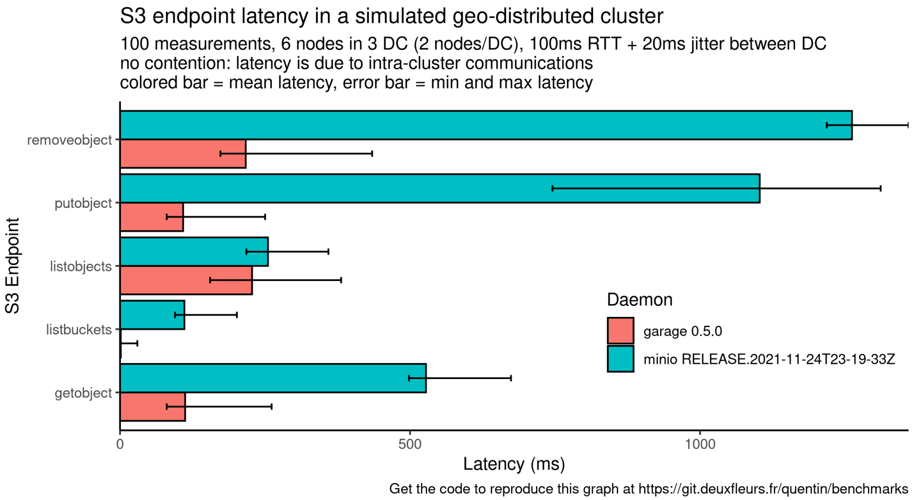
- **This API can be implemented using only weak primitives**
  (e.g. in the asynchronous message passing model with no further assumption)
  $\rightarrow$ the API is strictly weaker than consensus
  $\rightarrow$ we can implement it in Garage!

Why avoid consensus?

Consensus can be implemented reasonably well in practice, so why avoid it?

- **Software complexity:** RAFT and PAXOS are complex beasts;
  harder to prove, harder to reason about
- **Performance issues:**
  - Theoretical requirements (RNG, failure detector) translate into **practical costs**
  - The leader is a **bottleneck** for all requests;
    even in leaderless approaches, **all nodes must process all operations in order**
  - Particularly **sensitive to higher latency** between nodes

Performance gains in practice



What can we implement without consensus?

- Any **conflict-free replicated data type** (CRDT)

What can we implement without consensus?

- Any **conflict-free replicated data type** (CRDT)
- Non-transactional key-value stores such as S3 are equivalent to a simple CRDT: a map of **last-writer-wins registers** (each key is its own CRDT)

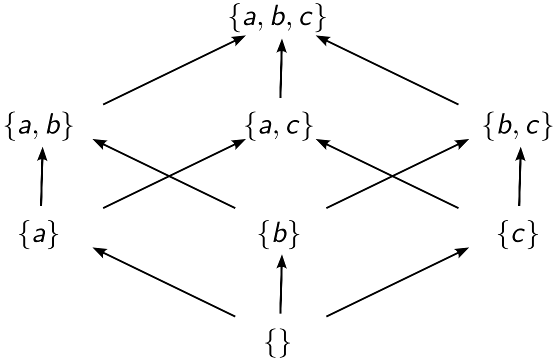
What can we implement without consensus?

- Any **conflict-free replicated data type** (CRDT)
- Non-transactional key-value stores such as S3 are equivalent to a simple CRDT: a map of **last-writer-wins registers** (each key is its own CRDT)
- **Read-after-write consistency** can be implemented using quorums on read and write operations

What can we implement without consensus?

- Any **conflict-free replicated data type** (CRDT)
- Non-transactional key-value stores such as S3 are equivalent to a simple CRDT: a map of **last-writer-wins registers** (each key is its own CRDT)
- **Read-after-write consistency** can be implemented using quorums on read and write operations
- **Monotonicity of reads** can be implemented with repair-on-read (makes reads more costly, not implemented in Garage)

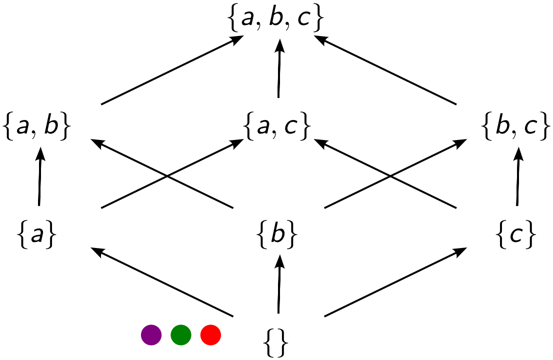
CRDTs and quorums: read-after-write consistency



CRDTs and quorums: read-after-write consistency

$write(\{a\})$:

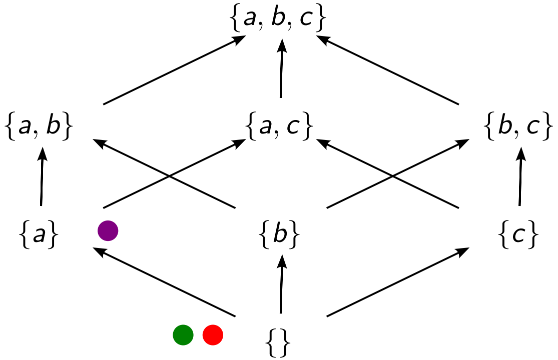
- ● $\not\supseteq \{a\}$
- ● $\not\supseteq \{a\}$
- ● $\not\supseteq \{a\}$



CRDTs and quorums: read-after-write consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

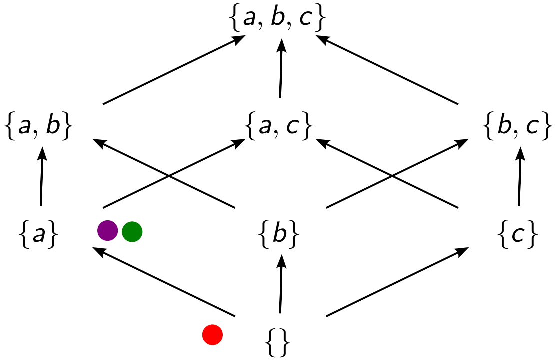


CRDTs and quorums: read-after-write consistency

$write(\{a\})$:

- 🟣 $\sqsupseteq \{a\} \to$ OK
- 🟢 $\sqsupseteq \{a\} \to$ OK
- 🔴 $\not\sqsupseteq \{a\}$

return OK



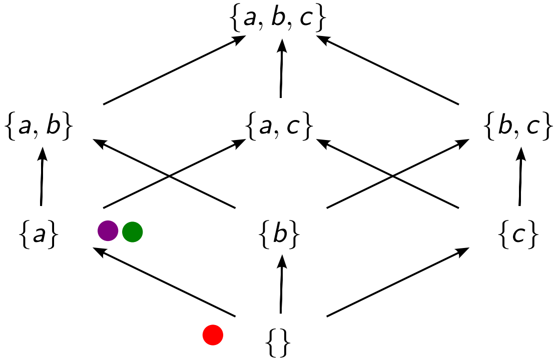
CRDTs and quorums: read-after-write consistency

write$(\{a\})$:

- (purple) $\sqsupseteq \{a\} \rightarrow$ OK
- (green) $\sqsupseteq \{a\} \rightarrow$ OK
- (red) $\not\sqsupseteq \{a\}$

return OK

read$()$:



CRDTs and quorums: read-after-write consistency

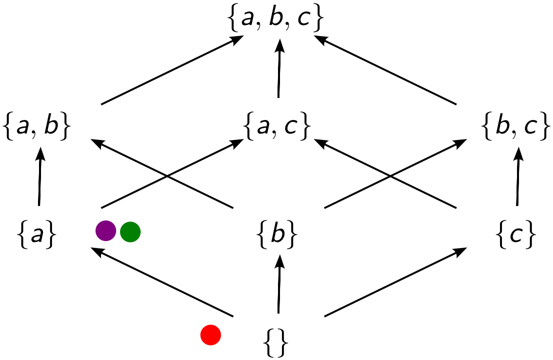
write($\{a\}$):

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$

return OK

read():

- ● $\rightarrow \{\}$



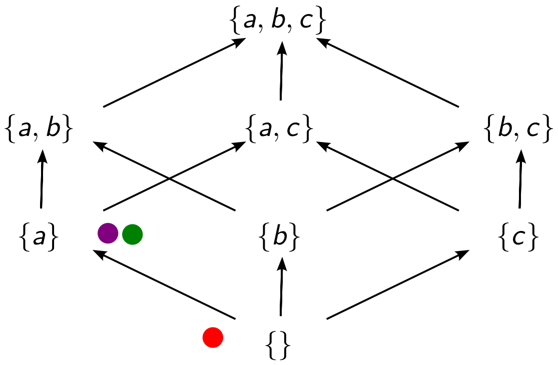
CRDTs and quorums: read-after-write consistency

write$(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- return OK

read$()$:

- ● $\rightarrow \{\}$
- ● $\rightarrow \{a\}$
- return $\{\} \sqcup \{a\} = \{a\}$



CRDTs and quorums: read-after-write consistency

write($\{a\}$):

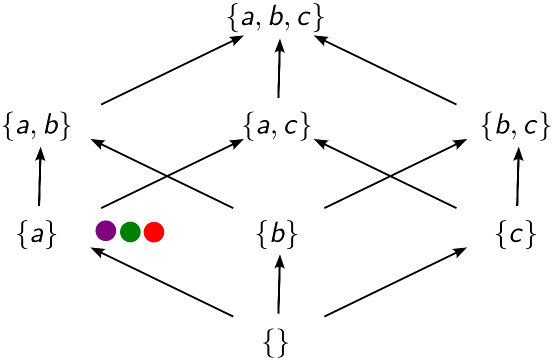
- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\sqsupseteq \{a\}$

return OK

read():

- ● $\rightarrow \{\}$
- ● $\rightarrow \{a\}$

return $\{\} \sqcup \{a\} = \{a\}$



CRDTs and quorums: read-after-write consistency

Property: If node A did an operation $write(x)$ and received an OK response,
and node B starts an operation $read()$ after A received OK,
then B will read a value $x' \sqsupseteq x$.

Algorithm $write(x)$**:**

1. Broadcast $write(x)$ to all nodes
2. Wait for $k > n/2$ nodes to reply OK
3. Return OK

Algorithm $read()$**:**

1. Broadcast $read()$ to all nodes
2. Wait for $k > n/2$ nodes to reply with values $x_1, \dots, x_k$
3. Return $x_1 \sqcup \dots \sqcup x_k$

Why does it work? There is at least one node at the intersection between the two sets of nodes that replied to each request, that "saw" x before the $read()$ started ($x_i \sqsupseteq x$).

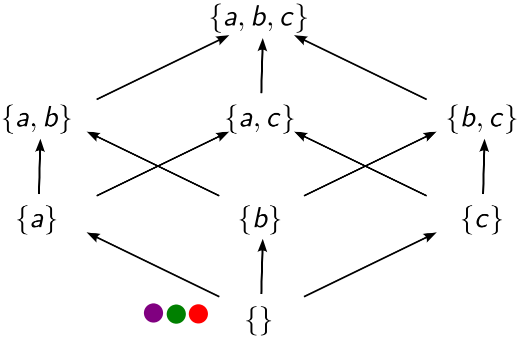
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- 🟣 $\not\supseteq \{a\}$
- 🟢 $\not\supseteq \{a\}$
- 🔴 $\not\supseteq \{a\}$

$write(\{b\})$:

- 🟣 $\not\supseteq \{b\}$
- 🟢 $\not\supseteq \{b\}$
- 🔴 $\not\supseteq \{b\}$



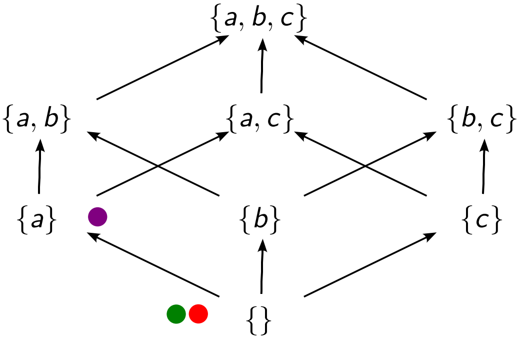
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \to$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\not\sqsupseteq \{b\}$
- ● $\not\sqsupseteq \{b\}$



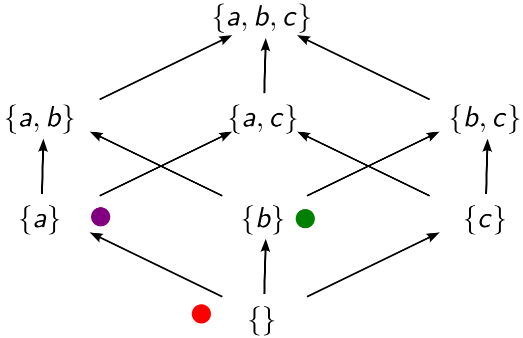
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- $\sqsupseteq \{a\} \rightarrow$ OK
- $\not\sqsupseteq \{a\}$
- $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- $\not\sqsupseteq \{b\}$
- $\sqsupseteq \{b\} \rightarrow$ OK
- $\not\sqsupseteq \{b\}$



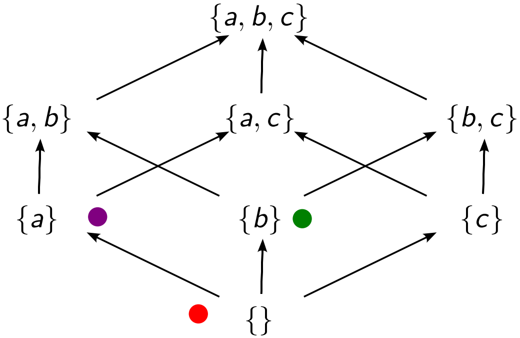
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



$read()$:

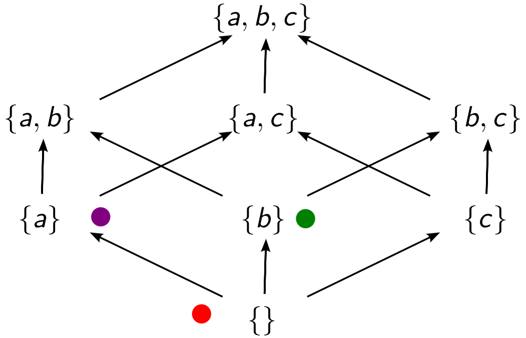
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- $\sqsupseteq \{a\} \rightarrow$ OK
- $\not\sqsupseteq \{a\}$
- $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- $\not\sqsupseteq \{b\}$
- $\sqsupseteq \{b\} \rightarrow$ OK
- $\not\sqsupseteq \{b\}$



$read()$:

- $\rightarrow \{a\}$

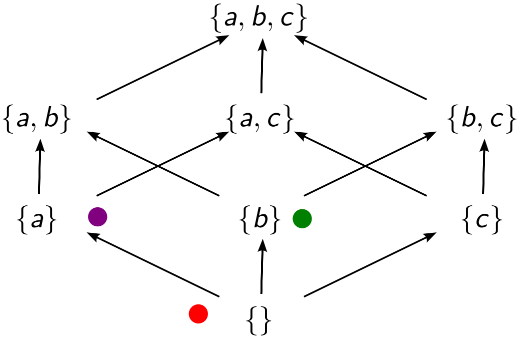
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



$read()$:

- ● $\rightarrow \{a\}$
- ● $\rightarrow \{\}$

return $\{a\}$

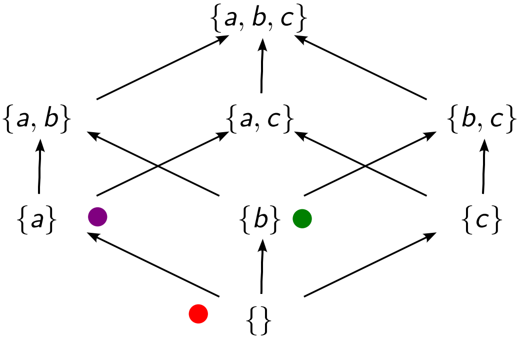
CRDTs and quorums: monotonic-reads consistency

write($\{a\}$):

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

write($\{b\}$):

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



read():

- ● $\rightarrow \{a\}$
- ● $\rightarrow \{\}$

return $\{a\}$

;

read():

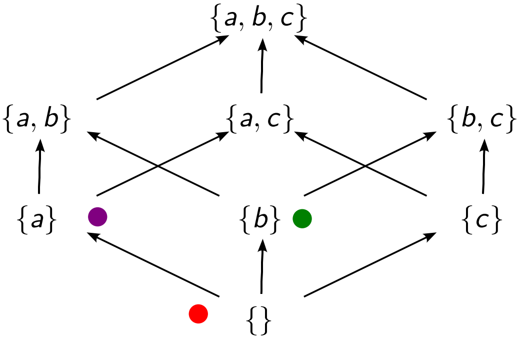
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



$read()$:

- ● $\rightarrow \{a\}$
- ● $\rightarrow \{\}$

return $\{a\}$

;

$read()$:

- ● $\rightarrow \{\}$

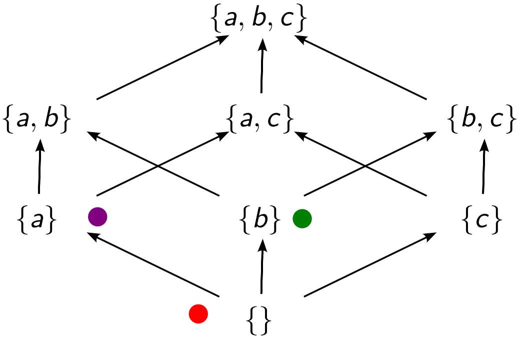
CRDTs and quorums: monotonic-reads consistency

$write(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

$write(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



$read()$:

- ● $\rightarrow \{a\}$
- ● $\rightarrow \{\}$

return $\{a\}$

;

$read()$:

- ● $\rightarrow \{\}$
- ● $\rightarrow \{b\}$

return $\{b\}$

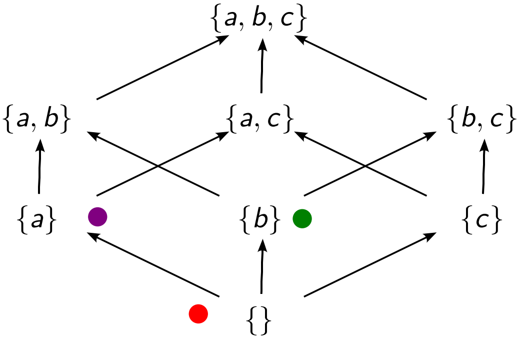
CRDTs and quorums: monotonic-reads consistency

write$(\{a\})$:

- ● $\sqsupseteq \{a\} \rightarrow$ OK
- ● $\not\sqsupseteq \{a\}$
- ● $\not\sqsupseteq \{a\}$

write$(\{b\})$:

- ● $\not\sqsupseteq \{b\}$
- ● $\sqsupseteq \{b\} \rightarrow$ OK
- ● $\not\sqsupseteq \{b\}$



read():

- ● $\rightarrow \{a\}$
- ● $\rightarrow \{\}$

return $\{a\}$

;

read():

- ● $\rightarrow \{\}$
- ● $\rightarrow \{b\}$

return $\{b\}$

??!

$\{a\} \not\sqsubseteq \{b\}$

CRDTs and quorums: monotonic-reads consistency

Property: If node A did an operation *read*() and received x as a response,
and node B starts an operation *read*() after A received x,
then B will read a value $x' \sqsupseteq x$.

Algorithm *monotonic_read*()**:** (a.k.a. repair-on-read)

1. Broadcast *read*() to all nodes
2. Wait for $k > n/2$ nodes to reply with values $x_1, \dots, x_k$
3. If $x_i \neq x_j$ for some nodes i and j,
   then call $write(x_1 \sqcup \dots \sqcup x_k)$ and wait for OK from $k' > n/2$ nodes
4. Return $x_1 \sqcup \dots \sqcup x_k$

This makes reads slower in some cases, and is **not implemented in Garage**.

A hard problem: layout changes

- We rely on quorums $k > n/2$ within each partition:

$$n = 3, \qquad k \geq 2$$

A hard problem: layout changes

- We rely on quorums $k > n/2$ within each partition:

$$n = 3, \qquad k \geq 2$$

- When rebalancing, the set of nodes responsible for a partition can change:

$$\{n_A, n_B, n_C\} \to \{n_A, n_D, n_E\}$$

A hard problem: layout changes

- We rely on quorums $k > n/2$ within each partition:

$$n = 3, \qquad k \geq 2$$

- When rebalancing, the set of nodes responsible for a partition can change:

$$\{n_A, n_B, n_C\} \to \{n_A, n_D, n_E\}$$

- During the rebalancing, D and E don't yet have the data,
  and B and C want to get rid of the data to free up space
  $\to$ quorums only within the new set of nodes don't work
  $\to$ how to coordinate? **currently, we don't...**

Operating big Garage clusters

Operating Garage

```
[root@celeri:~]# docker exec -ti 74a09 /garage status
==== HEALTHY NODES ====
ID                Hostname   Address                                     Tags                                Zone     Capacity  DataAvail
a717e5b618267806  courgette  [2001:910:1204:1::32]:3901                  [courgette,neptune,france,alex]     neptune  5         393.3 GB (78.7%)
8cf284e7df17d0fd  celeri     [2001:910:1204:1::33]:3901                  [celeri,neptune,france,alex]        neptune  20        1.6 TB (78.3%)
0a03ab7c082ad929  ananas     [2a01:e0a:e4:2dd0::42]:3901                 [ananas,scorpio,france,adrien]      scorpio  20        1.7 TB (83.6%)
fdfaf7832d8359e0  df-ymk     [2a02:a03f:6510:5102:6e4b:90ff:fe3b:e939]:3901  [df-ymk,bespin,belgium,max]      bespin   4         263.3 GB (52.7%)
5fcb3b6e39db3dcb  concombre  [2001:910:1204:1::31]:3901                  [concombre,neptune,france,alex]     neptune  5         393.4 GB (78.7%)
942dd71ea95f4904  df-ymf     [2a02:a03f:6510:5102:6e4b:90ff:fe3a:6174]:3901  [df-ymf,bespin,belgium,max]      bespin   4         264.5 GB (52.9%)
17ee03c6b81d9235  df-ykl     [2a02:a03f:6510:5102:6e4b:90ff:fe3b:e86c]:3901  [df-ykl,bespin,belgium,max]      bespin   4         280.2 GB (56.1%)
2032d0a37f249c4a  abricot    [2a01:e0a:e4:2dd0::41]:3901                 [abricot,scopio,france,adrien]      scorpio  20        1.7 TB (83.7%)
```

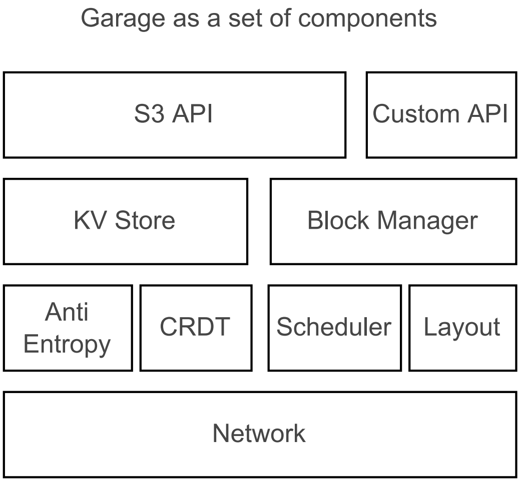
Operating Garage

```
[root@celeri:~]# docker exec -ti 74a09 /garage status
==== HEALTHY NODES ====
ID                Hostname   Address                                     Tags                               Zone     Capacity  DataAvail
a717e5b618267806  courgette  [2001:910:1204:1::32]:3901                  [courgette,neptune,france,alex]    neptune  5         393.3 GB (78.7%)
8cf284e7df17d0fd  celeri     [2001:910:1204:1::33]:3901                  [celeri,neptune,france,alex]       neptune  20        1.6 TB (78.3%)
0a03ab7c082ad929  ananas     [2a01:e0a:e4:2dd0::42]:3901                 [ananas,scorpio,france,adrien]     scorpio  20        1.7 TB (83.6%)
fdfaf7832d8359e0  df-ymk     [2a02:a03f:6510:5102:6e4b:90ff:fe3b:e939]:3901  [df-ymk,bespin,belgium,max]    bespin   4         263.3 GB (52.7%)
5fcb3b6e39db3dcb  concombre  [2001:910:1204:1::31]:3901                  [concombre,neptune,france,alex]    neptune  5         393.4 GB (78.7%)
942dd71ea95f4904  df-ymf     [2a02:a03f:6510:5102:6e4b:90ff:fe3a:6174]:3901  [df-ymf,bespin,belgium,max]    bespin   4         264.5 GB (52.9%)
17ee03c6b81d9235  df-ykl     [2a02:a03f:6510:5102:6e4b:90ff:fe3b:e86c]:3901  [df-ykl,bespin,belgium,max]    bespin   4         280.2 GB (56.1%)
2032d0a37f249c4a  abricot    [2a01:e0a:e4:2dd0::41]:3901                 [abricot,scopio,france,adrien]     scorpio  20        1.7 TB (83.7%)
```

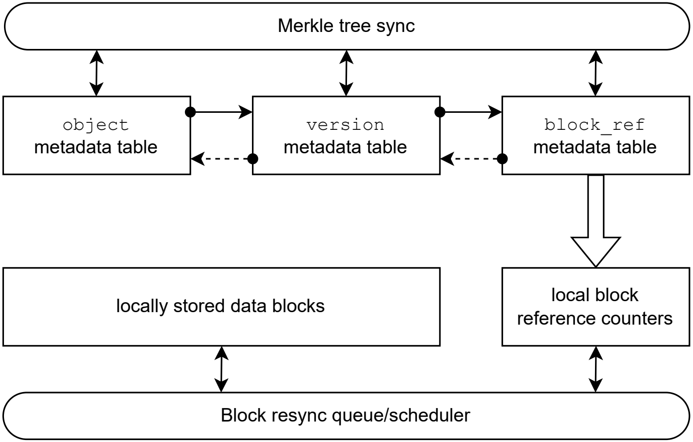
```
[lx@lindy:~]$ nomad exec -job garage-staging garage status
==== HEALTHY NODES ====
ID                Hostname  Address                                      Tags        Zone     Capacity  DataAvail
967786691f20bb79  caribou   [2001:910:1204:1::23]:3991                   [caribou]   neptune  5         450.5 GB (91.8%)
ec5753c546756825  df-pw5    [2a02:a03f:6510:5102:223:24ff:feb0:e8a7]:3991  [df-pw5]  bespin   5         436.7 GB (90.6%)
3aed398eec82972b  origan    [2a01:e0a:5e4:1d0:223:24ff:feaf:fdec]:3991   [origan]    jupiter  5         461.9 GB (94.1%)
76797283f6c7e162  carcajou  [2001:910:1204:1::22]:3991                   [carcajou]  neptune  2         165.4 GB (73.1%)

==== FAILED NODES ====
ID                Hostname  Address  Tags       Zone    Capacity  Last seen
8073f25ffb7d6944  ??        ??       [piranha]  corrin  gateway   never seen
```

Garage's architecture



Garage's architecture



Digging deeper

```
[root@celeri:~]# docker exec -ti 74a09 /garage stats

Garage version: v0.8.2 [features: k2v, sled, lmdb, sqlite, consul-discovery, kubernetes-discovery, metrics, telemetry-otlp, bundled-libs]
Rust compiler version: 1.63.0

Database engine: LMDB (using Heed crate)

Table stats:
  Table      Items    MklItems  MklTodo  GcTodo
  bucket_v2  124      150       0        0
  key        56       59        0        0
  object     607630   749155    0        0
  version    498551   553360    0        1285
  block_ref  1098024  1192560   0        470

Block manager stats:
  number of RC entries (~= number of blocks): 594820
  resync queue length: 3
  blocks with resync errors: 1

If values are missing above (marked as NC), consider adding the --detailed flag (this will be slow).

Storage nodes:
  ID                Hostname   Zone     Capacity  Part.  DataAvail                   MetaAvail
  942dd71ea95f4904  df-ymf     bespin   4         86     264.5 GB/499.9 GB (52.9%)  264.5 GB/499.9 GB (52.9%)
  a717e5b618267806  courgette  neptune  5         42     393.3 GB/499.9 GB (78.7%)  372.8 GB/486.4 GB (76.7%)
  17ee03c6b81d9235  df-ykl     bespin   4         85     280.2 GB/499.9 GB (56.1%)  280.2 GB/499.9 GB (56.1%)
  5fcb3b6e39db3dcb  concombre  neptune  5         42     393.4 GB/499.9 GB (78.7%)  380.4 GB/486.4 GB (78.2%)
  fdfaf7832d8359e0  df-ymk     bespin   4         85     263.3 GB/499.9 GB (52.7%)  263.3 GB/499.9 GB (52.7%)
  0a03ab7c082ad929  ananas     scorpio  20        128    1.7 TB/2.0 TB (83.6%)      396.2 GB/477.9 GB (82.9%)
  8cf284e7df17d0fd  celeri     neptune  20        172    1.6 TB/2.0 TB (78.3%)      417.3 GB/486.4 GB (85.8%)
  2032d0a37f249c4a  abricot    scorpio  20        128    1.7 TB/2.0 TB (83.7%)      433.2 GB/482.7 GB (89.7%)

Estimated available storage space cluster-wide (might be lower in practice):
  data: 787.4 GB
  metadata: 621.1 GB
```

Digging deeper

```
[root@celeri:~]# docker exec -ti 74a09 /garage worker list
TID  State  Name                          Tranq  Done  Queue  Errors  Consec  Last
1    Idle   Block resync worker #1        1      -     3      -       -
2    Idle   Block resync worker #2        1      -     3      -       -
3    Idle   Block resync worker #3        -      -     -      -       -
4    Idle   Block resync worker #4        -      -     -      -       -
5    Idle   Block scrub worker            4      -     -      -       -
6    Idle   bucket_v2 Merkle              -      -     0      -       -
7    Idle   bucket_v2 sync                -      -     0      -       -
8    Idle   bucket_v2 GC                  -      -     0      -       -
9    Idle   bucket_v2 queue               -      -     0      -       -
10   Idle   bucket_alias Merkle           -      -     0      -       -
11   Idle   bucket_alias sync             -      -     0      -       -
12   Idle   bucket_alias GC               -      -     0      -       -
13   Idle   bucket_alias queue            -      -     0      -       -
14   Idle   key Merkle                    -      -     0      -       -
15   Idle   key sync                      -      -     0      -       -
16   Idle   key GC                        -      -     0      -       -
17   Idle   key queue                     -      -     0      -       -
18   Idle   object Merkle                 -      -     0      -       -
19   Idle   object sync                   -      -     0      -       -
20   Idle   object GC                     -      -     0      -       -
21   Idle   object queue                  -      -     0      -       -
22   Idle   bucket_object_counter Merkle  -      -     0      -       -
23   Idle   bucket_object_counter sync    -      -     0      -       -
24   Idle   bucket_object_counter GC      -      -     0      -       -
25   Idle   bucket_object_counter queue   -      -     0      4       0       3 days ago
26   Idle   version Merkle                -      -     0      -       -
27   Idle   version sync                  -      -     0      -       -
28   Idle   version GC                    -      -     1285   -       -
29   Idle   version queue                 -      -     0      -       -
30   Idle   block_ref Merkle              -      -     0      -       -
31   Idle   block_ref sync                -      -     0      -       -
32   Idle   block_ref GC                  -      -     470    -       -
33   Idle   block_ref queue               -      -     0      -       -
```

Digging deeper

```
[root@celeri:~]# docker exec -ti 74a09 /garage worker get
8cf284e7df17d0fd  resync-tranquility         1
8cf284e7df17d0fd  resync-worker-count        2
8cf284e7df17d0fd  scrub-corruptions_detected  0
8cf284e7df17d0fd  scrub-last-completed       2023-09-09T19:10:37.167Z
8cf284e7df17d0fd  scrub-next-run             2023-10-07T05:51:49.167Z
8cf284e7df17d0fd  scrub-tranquility          4

[root@celeri:~]# docker exec -ti 74a09 /garage worker get -a resync-tranquility
0a03ab7c082ad929  resync-tranquility  1
17ee03c6b81d9235  resync-tranquility  1
2032d0a37f249c4a  resync-tranquility  1
5fcb3b6e39db3dcb  resync-tranquility  1
8cf284e7df17d0fd  resync-tranquility  1
942dd71ea95f4904  resync-tranquility  1
a717e5b618267806  resync-tranquility  1
fdfaf7832d8359e0  resync-tranquility  1
```

Potential limitations and bottlenecks

- Global:
  - Max. ~100 nodes per cluster (excluding gateways)
- Metadata:
  - One big bucket = bottleneck, object list on 3 nodes only
- Block manager:
  - Lots of small files on disk
  - Processing the resync queue can be slow
  - Multi-HDD support not yet released (soon!)

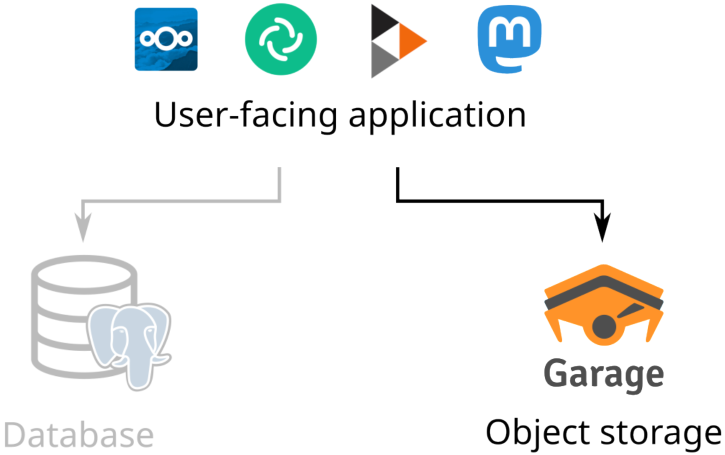
Deployment advice for very large clusters

- Metadata storage:
  - ZFS mirror (x2) on fast NVMe
  - Use LMDB storage engine
- Data block storage:
  - Wait for v0.9 with multi-HDD support
  - XFS on individual drives
  - Increase block size (1MB $\rightarrow$ 10MB, requires more RAM and good networking)
  - Tune `resync-tranquility` and `resync-worker-count` dynamically
- Other :
  - Split data over several buckets
  - Use less than 100 storage nodes
  - Use gateway nodes

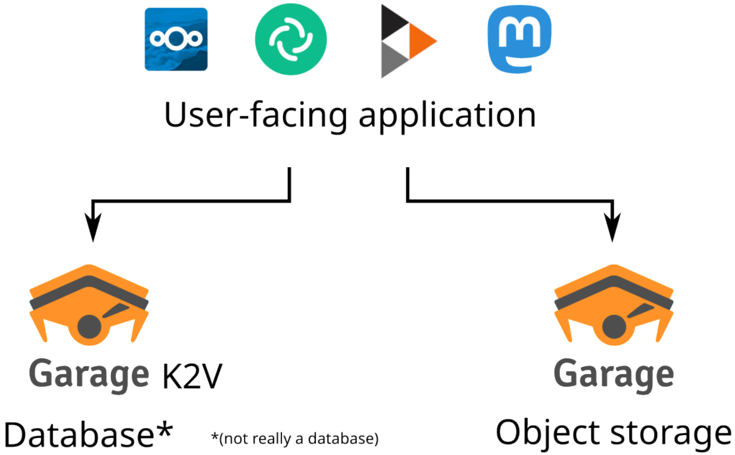
Current deployments: $<$ 10 TB, we don't have much experience with more

Going further than the S3 API

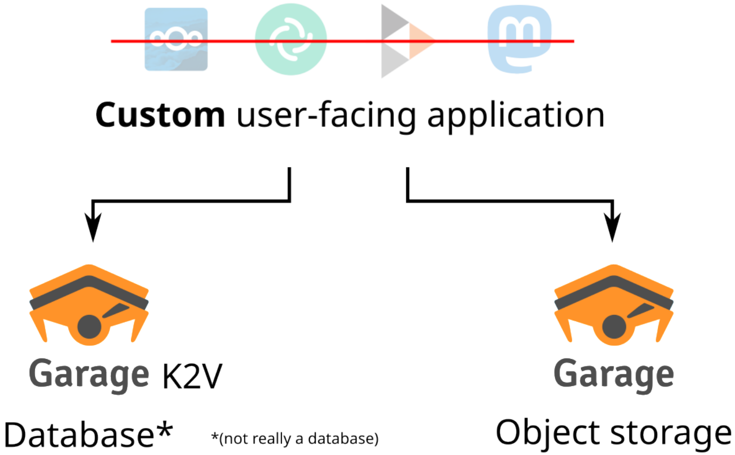
Using Garage for everything



Using Garage for everything



Using Garage for everything



K2V Design

- A new, custom, minimal API
  - Single-item operations
  - Operations on ranges and batches of items
  - Polling operations to help implement a PubSub pattern

K2V Design

- A new, custom, minimal API
  - Single-item operations
  - Operations on ranges and batches of items
  - Polling operations to help implement a PubSub pattern
- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)

K2V Design

- A new, custom, minimal API
  - Single-item operations
  - Operations on ranges and batches of items
  - Polling operations to help implement a PubSub pattern

- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)

- Weakly consistent, CRDT-friendly
  $\rightarrow$ no support for transactions (not ACID)

K2V Design

- A new, custom, minimal API
  - Single-item operations
  - Operations on ranges and batches of items
  - Polling operations to help implement a PubSub pattern
- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)
- Weakly consistent, CRDT-friendly
  → no support for transactions (not ACID)
- Cryptography-friendly: values are binary blobs

Handling concurrent values

How to handle concurrency? Example:

1. Client A reads the initial value of a key, x_0

Handling concurrent values

How to handle concurrency? Example:

1. Client A reads the initial value of a key, x_0
2. Client B also reads the initial value x_0 of that key

Handling concurrent values

How to handle concurrency? Example:

1. Client A reads the initial value of a key, x_0
2. Client B also reads the initial value x_0 of that key
3. Client A modifies x_0, and writes a new value x_1

Handling concurrent values

How to handle concurrency? Example:

1. Client A reads the initial value of a key, x_0
2. Client B also reads the initial value x_0 of that key
3. Client A modifies x_0, and writes a new value x_1
4. Client B also modifies x_0, and writes a new value x'_1, without having a chance to first read x_1

   $\rightarrow$ what should the final state be?

Handling concurrent values

- If we keep only x_1 or x_1', we risk **loosing application data**

Handling concurrent values

- If we keep only x_1 or x'_1, we risk **loosing application data**
- Values are opaque binary blobs, **K2V cannot resolve conflicts** by itself (e.g. by implementing a CRDT)

Handling concurrent values

- If we keep only x_1 or x'_1, we risk **loosing application data**
- Values are opaque binary blobs, **K2V cannot resolve conflicts** by itself (e.g. by implementing a CRDT)
- Solution: **we keep both!**
  $\rightarrow$ the value of the key is now $\{x_1, x'_1\}$
  $\rightarrow$ the client application can decide how to resolve conflicts on the next read

Keeping track of causality

How does K2V know that x_1 and x'_1 are concurrent?

- *read*() returns **a set of values** and an associated **causality token**

Keeping track of causality

How does K2V know that x_1 and x_1' are concurrent?

- *read*() returns **a set of values** and an associated **causality token**
- When calling *write*(), the client sends **the causality token from its last read**

Keeping track of causality

How does K2V know that x_1 and x'_1 are concurrent?

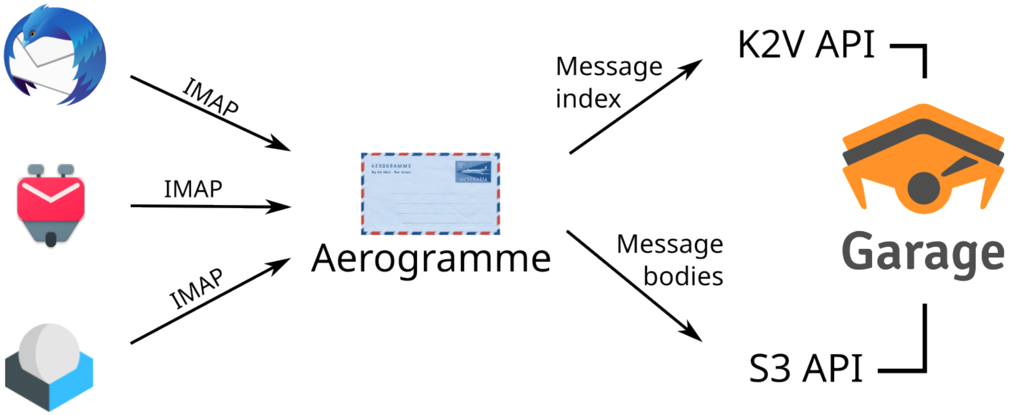
- *read*() returns **a set of values** and an associated **causality token**
- When calling *write*(), the client sends **the causality token from its last read**
- The causality token represents the set of values **already seen by the client**
  $\rightarrow$ those values are the **causal past** of the write operation
  $\rightarrow$ K2V can keep concurrent values and overwrite all ones in the causal past

Keeping track of causality

How does K2V know that x_1 and x_1' are concurrent?

- *read*() returns **a set of values** and an associated **causality token**
- When calling *write*(), the client sends **the causality token from its last read**
- The causality token represents the set of values **already seen by the client**
  $\rightarrow$ those values are the **causal past** of the write operation
  $\rightarrow$ K2V can keep concurrent values and overwrite all ones in the causal past
- Internally, the causality token is **a vector clock**

Application: an e-mail storage server



Aerogramme data model

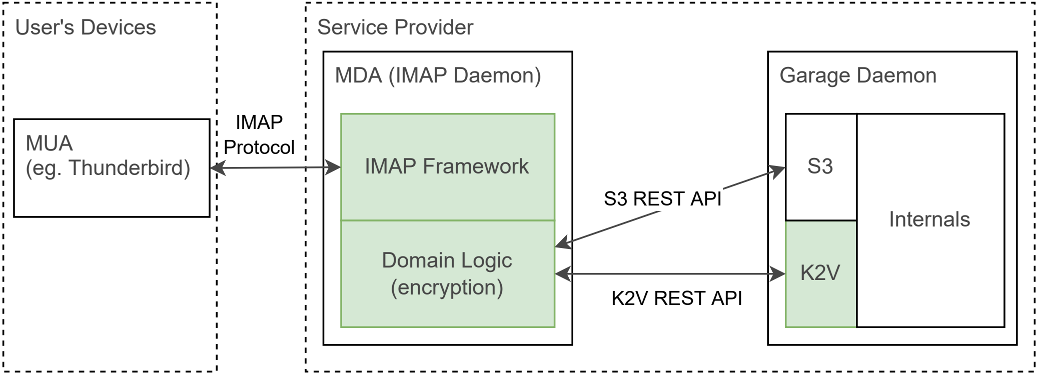
	immutable	**mutable**
K2V	Email Summary	Log
S3	Full Email	Checkpoint

Aerogramme data model

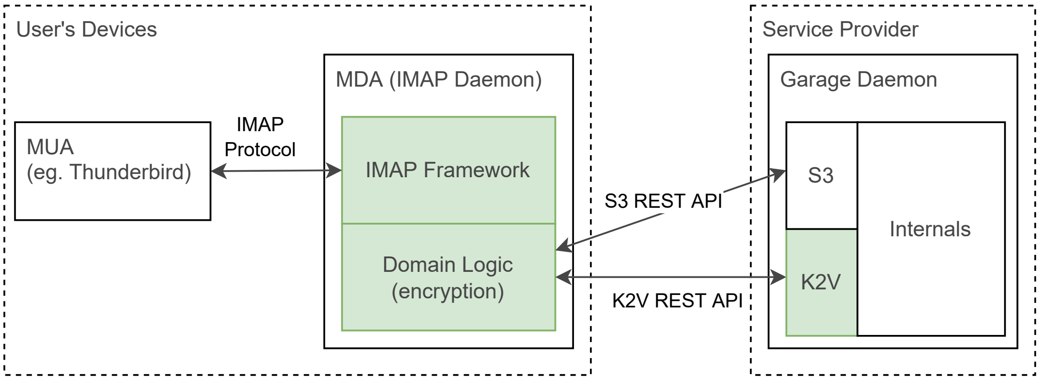
	immutable	**mutable**
K2V	Email Summary	Log
S3	Full Email	Checkpoint

Aerogramme encrypts all stored values for privacy
(Garage server administrators can't read your mail)

Different deployment scenarios



Different deployment scenarios



A new model for building resilient software

How to build an application using only Garage as a data store:

1. Design a data model suited to K2V
   (see Cassandra docs on porting SQL data models to Cassandra)
   - Use CRDTs or other eventually consistent data types (see e.g. Bayou)
   - Store opaque binary blobs to provide End-to-End Encryption

A new model for building resilient software

How to build an application using only Garage as a data store:

1. Design a data model suited to K2V
   (see Cassandra docs on porting SQL data models to Cassandra)
   - Use CRDTs or other eventually consistent data types (see e.g. Bayou)
   - Store opaque binary blobs to provide End-to-End Encryption
2. Store big blobs (files) using the S3 API

A new model for building resilient software

How to build an application using only Garage as a data store:

1. Design a data model suited to K2V
   (see Cassandra docs on porting SQL data models to Cassandra)
   - Use CRDTs or other eventually consistent data types (see e.g. Bayou)
   - Store opaque binary blobs to provide End-to-End Encryption
2. Store big blobs (files) using the S3 API
3. Let Garage manage sharding, replication, failover, etc.

Conclusion

Perspectives

- Fix the consistency issue when rebalancing
- Write about Garage's architecture and properties,
  and about our proposed architecture for (E2EE) apps over K2V+S3
- Continue developing Garage; finish Aerogramme; build new applications...
- Anything else?

Where to find us

Garage

`https://garagehq.deuxfleurs.fr/`

`mailto:garagehq@deuxfleurs.fr`

`#garage:deuxfleurs.fr` on Matrix

Free Software

Free as in Freedom