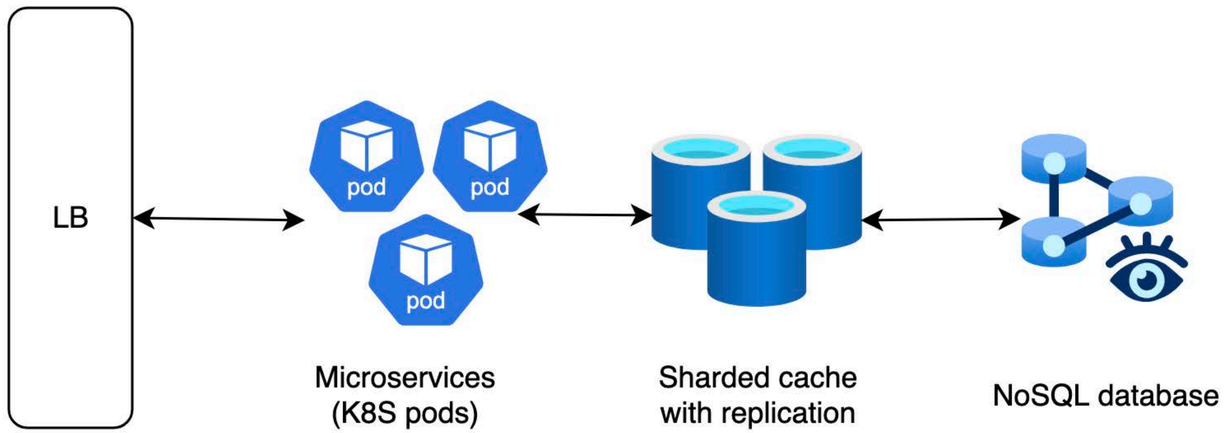
Designing a Read-Intensive Architecture for Information Systems with High Data Throughput

Building high-load systems isn’t just adding resources. Efficient storage, caching, and network optimization are key, while well-designed data schemes and architectural decisions ensure performance and SLA compliance at scale

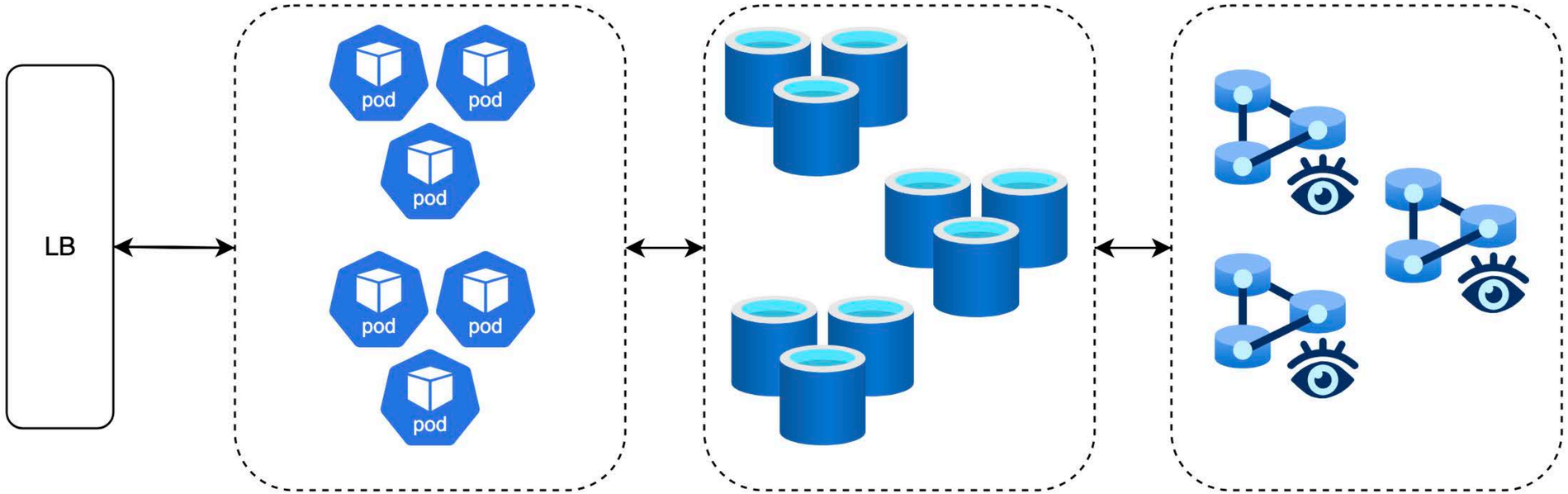
Ivan Sinitsin

What does a typical Read-Intensive system look like

- K8S
- Microservices
- Sharded cache with replication
- Sharded NoSQL DB with replication



When "Just Add Resources" Breaks

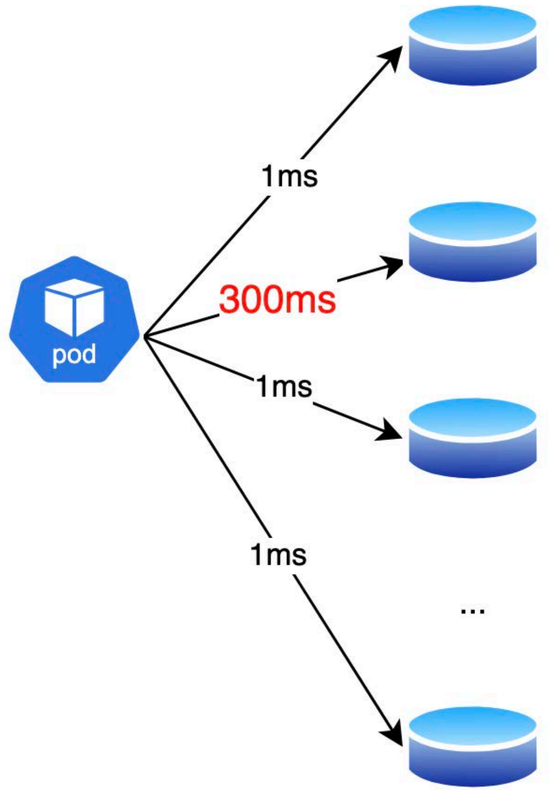


- Tail Latency (p99)
- Entropy of protocols (Gossip)
- Contention (The struggle for resources)
- Throughput vs Latency

- High Cardinality and Drift Topology

Tail Latency

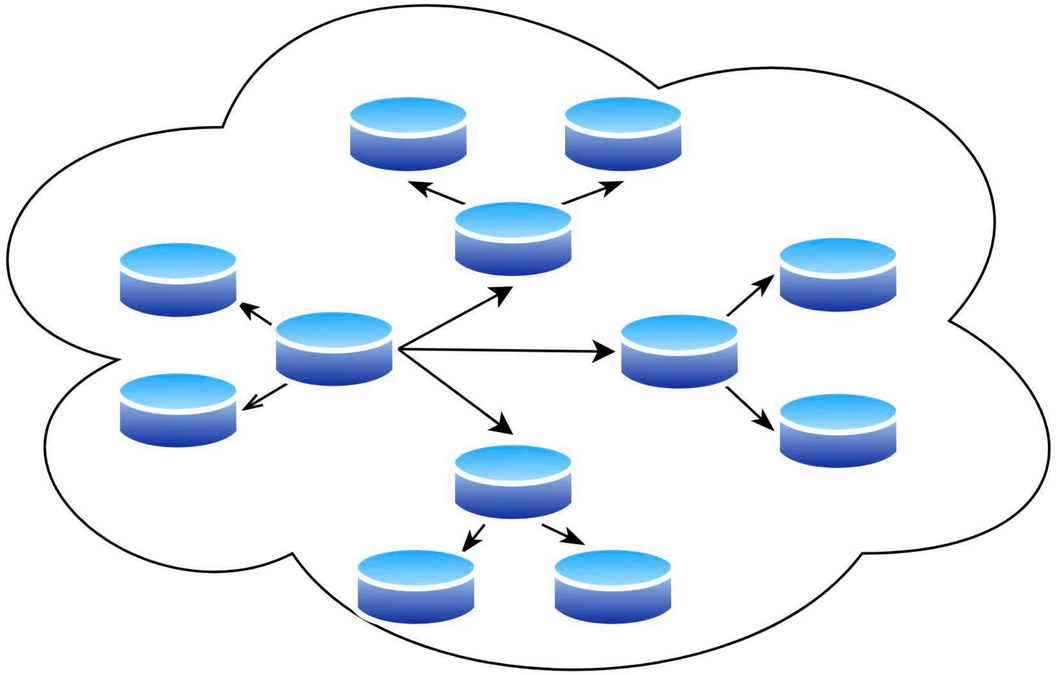
- Fan-out scheme: one API request throttles 50 shards
- Out of 1,000 nodes, the probability of encountering a slow container is 100%. One slow shard slows down the entire response.



Entropy of protocols

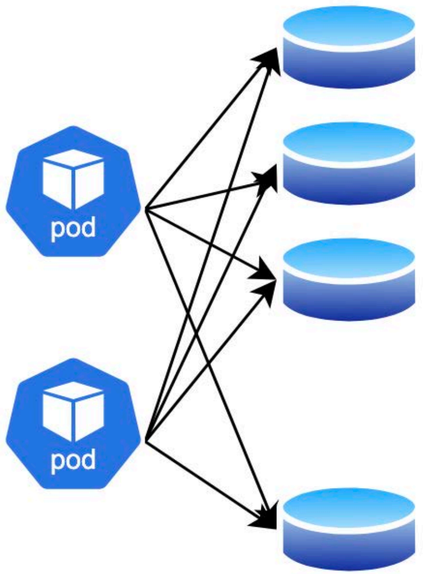
Scalability Tax: Increased overhead with increasing number of nodes

- Orchestration
- CPU
- Network traffic

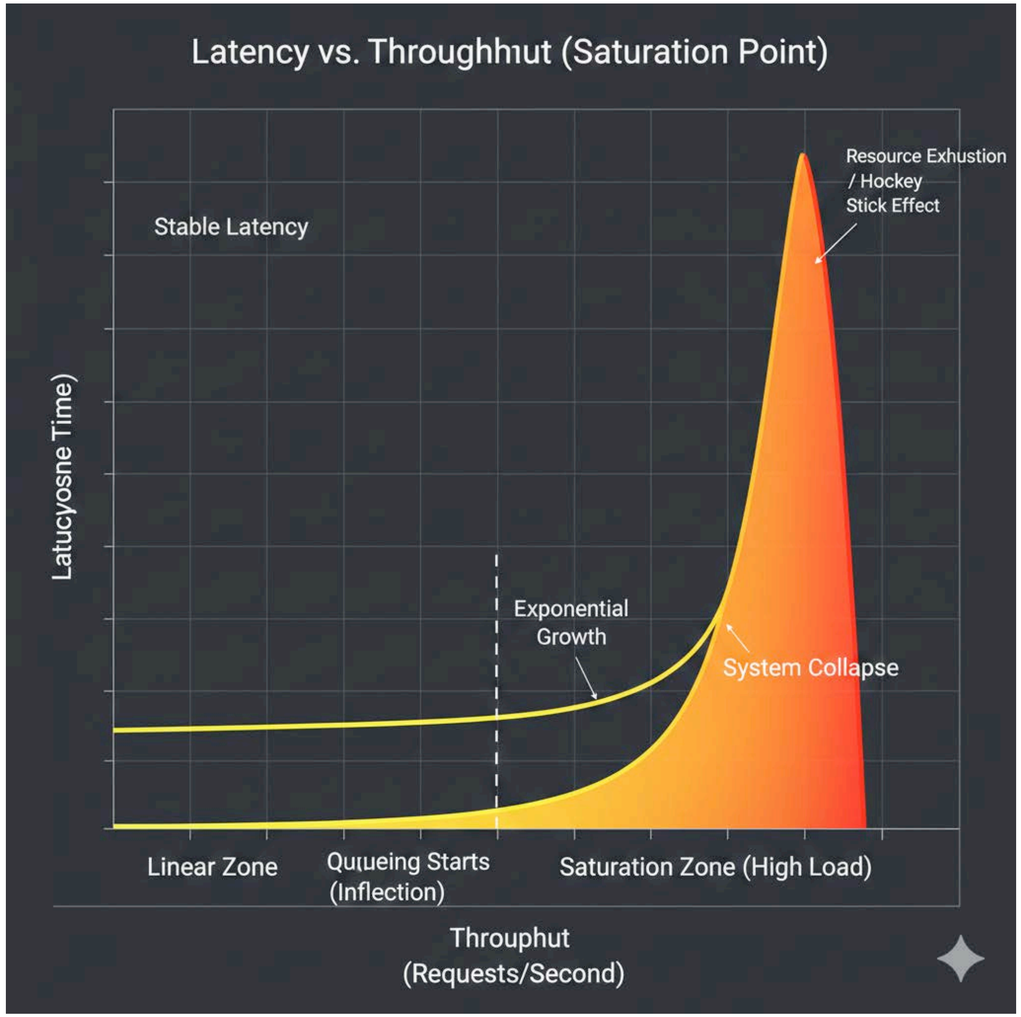


Contention & Resource Exhaustion

- N service pods
- M cache pods
- M * N connecitons
- N * M descriptors
- …



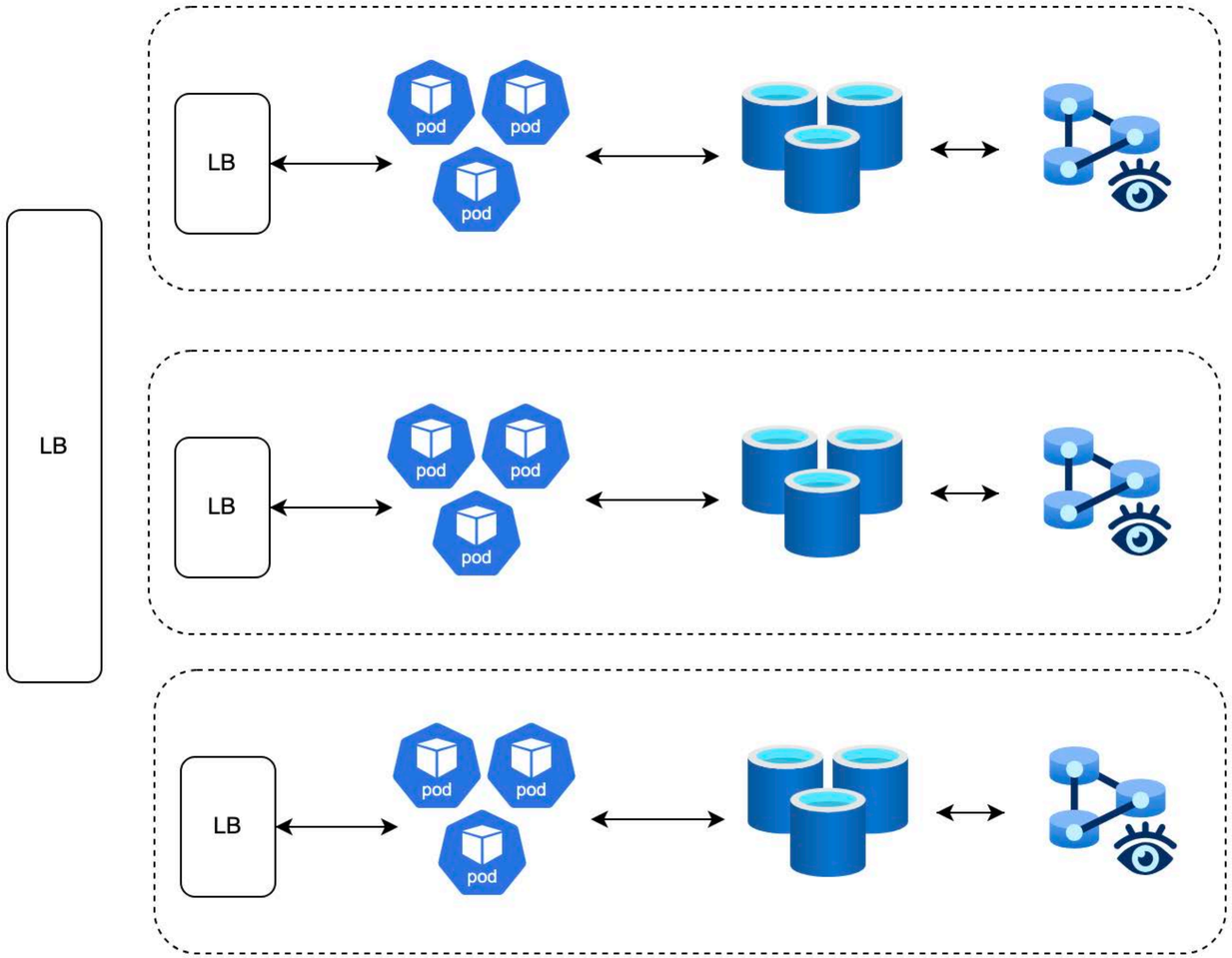
Throughput vs Latency



Defense Line: "Painkillers" for Highload

- Graceful Degradation: A cache crash doesn't equal a service crash (enable Circuit Breaker)
- Soft Timeouts: Don't wait forever for database. 200ms—and we'll serve what we have
- Stale-data: "Expired" is better than a 500 error

Zonal isolation (Multi-AZ) and its price



The Illusion of Stability: Why Symptoms Are Hidden, But the Disease Remains

- Hidden Tails: Your Soft Timeouts simply "truncate" the graph, turning real database bottlenecks into "old data" for the user
- Entropy of protocols: In Multi-AZ, Database service traffic between zones is growing exponentially
- Resource Death: 1,000 pods still hold 50,000 connections to Cache. Circuit breakers do not reduce the socket count

Optimization Strategy: Where are we losing resources?

Process data

Inefficient data format and schema

Storing data

Inefficient data format and schema

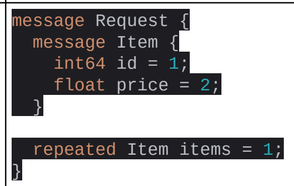
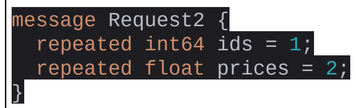
Transfer data

Inefficient data formats
Poor understanding of data format encoding schemes

Storage: More than just bytes

- JSON: Simple, but it's a text format. Expensive to parse, redundant
- Protobuf: Binary standard. Compact, strict, efficient
- FlatBuffers: Zero-copy access. Read only what's needed, without deserializing the entire object
- Columnar approach (Avro/Parquet): Read only the required fields from wide data structures
- Schema Registry: Extract metadata from the message body

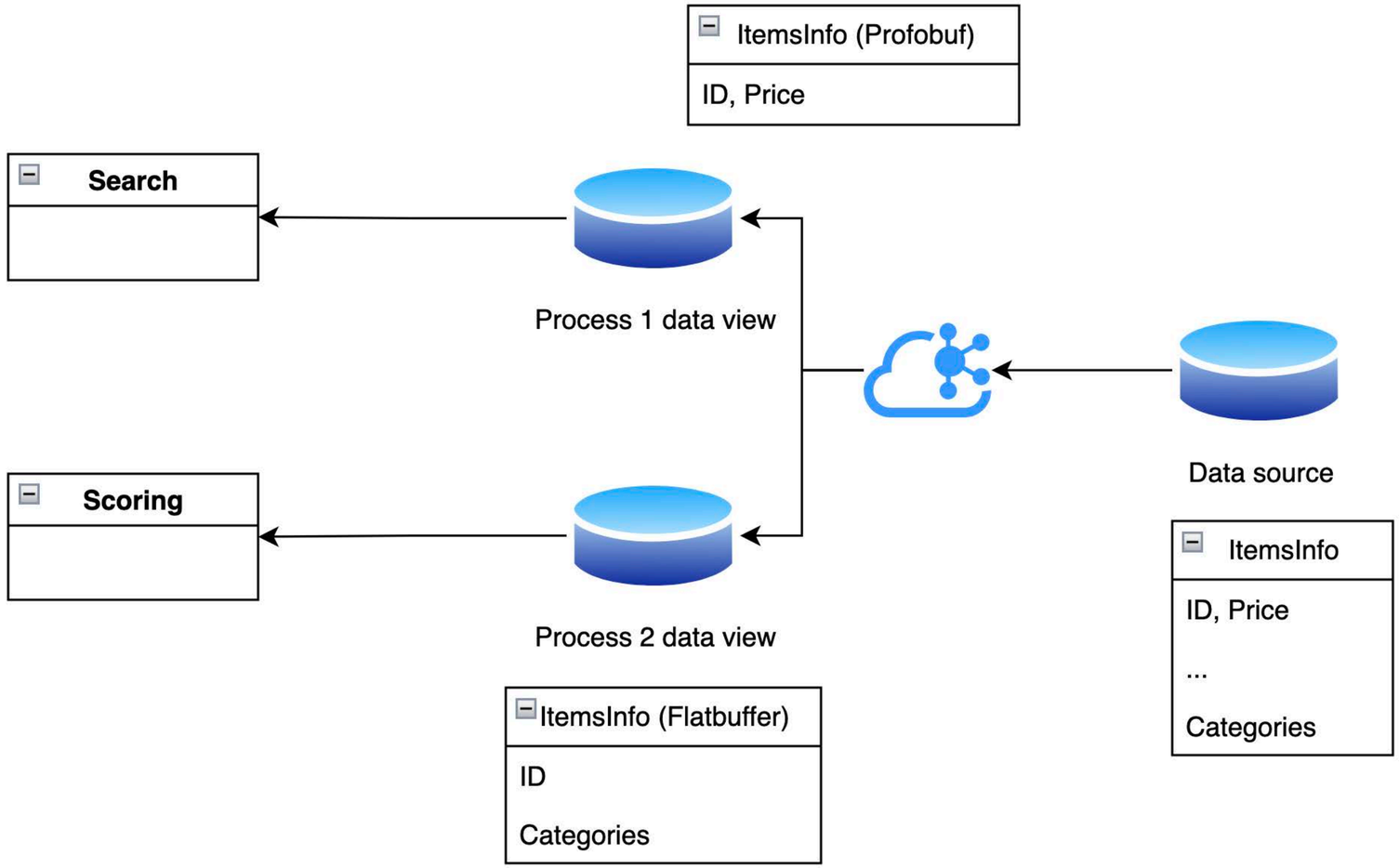
When convenience gets in the way of performance

	AoS (Array of Structures)	SoA (Structure of Arrays)
schema		
use	easy	hard
aggregation	poor memory locality	good memory locality
runtime effects	- large number of objects in heap - slow serialization	- low number of object in heap - fast serialization
SIMD	no	yes

Architectural Optimization: “CQRS” and Prepared Data

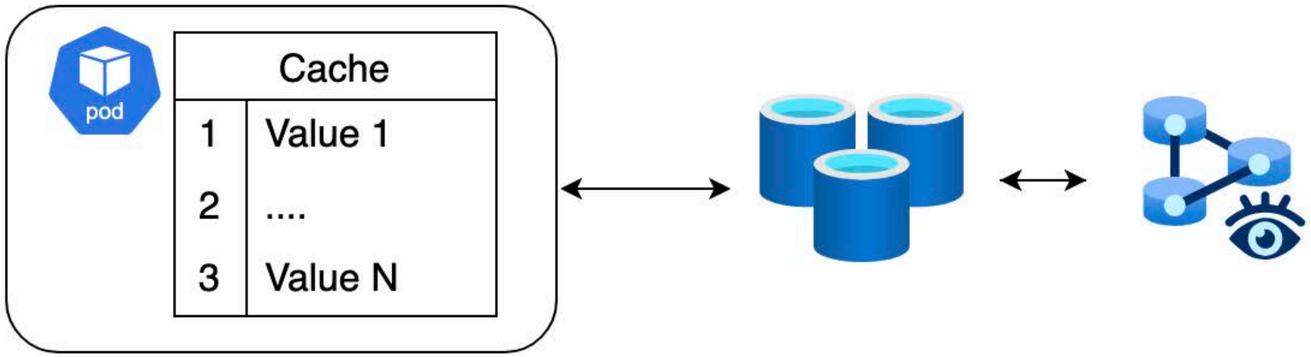
Use process-efficient

- Data format
- Data schema
- Data storage



Caching at your fingertips

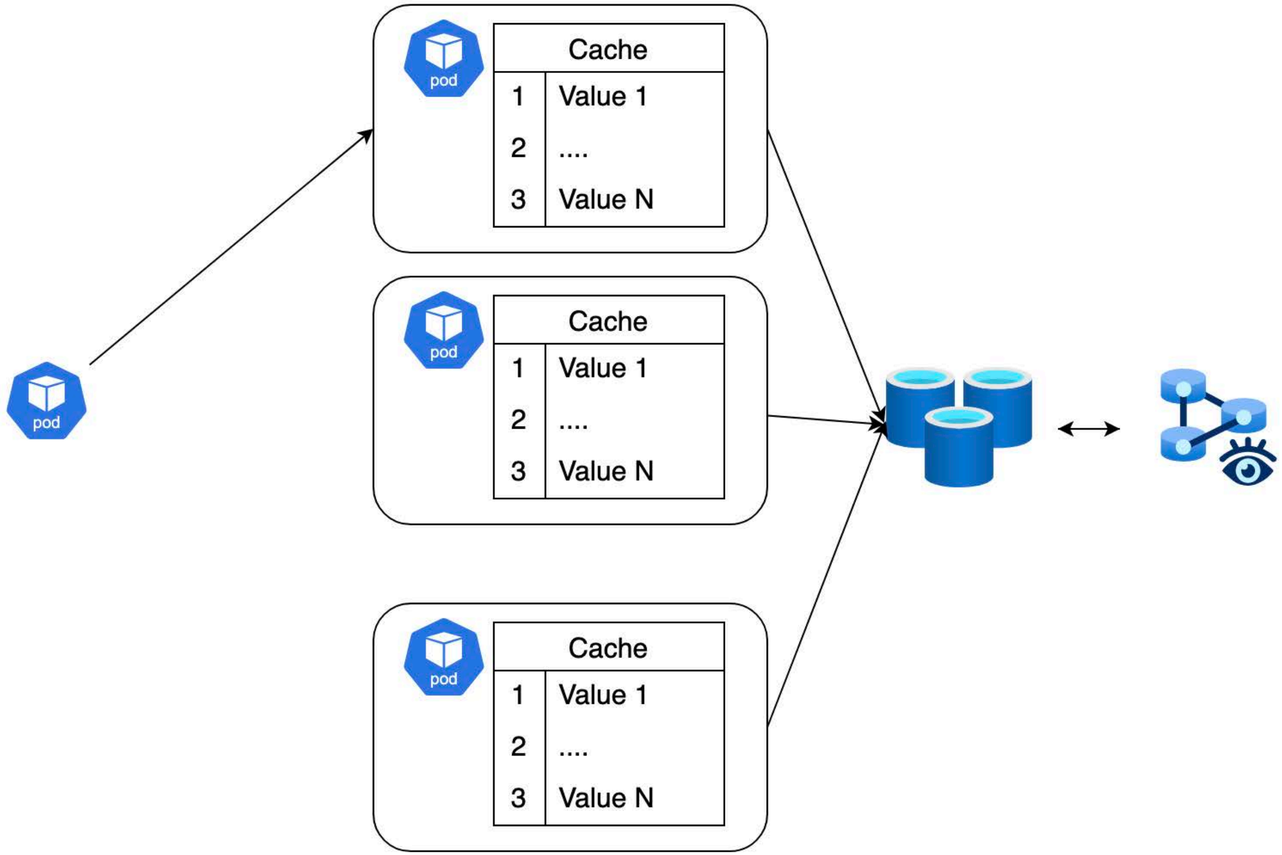
Add K8S-pod caching



Pod sharding

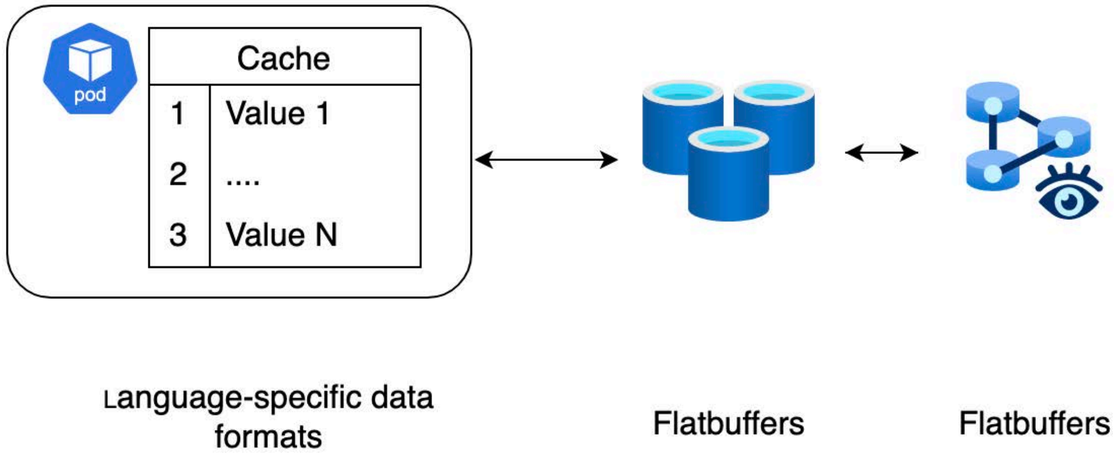
Use

- Service discovery
- Client-side balancing
- Consistent / Rendezvous hashing



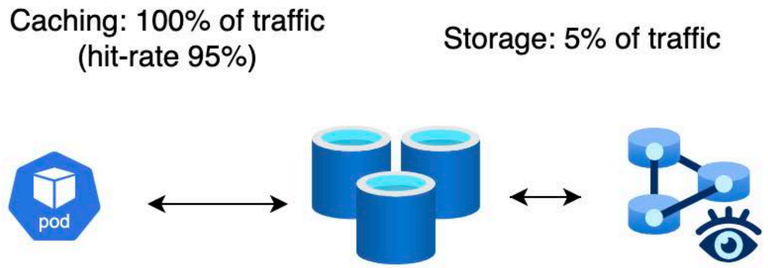
Choose different formats for different storages

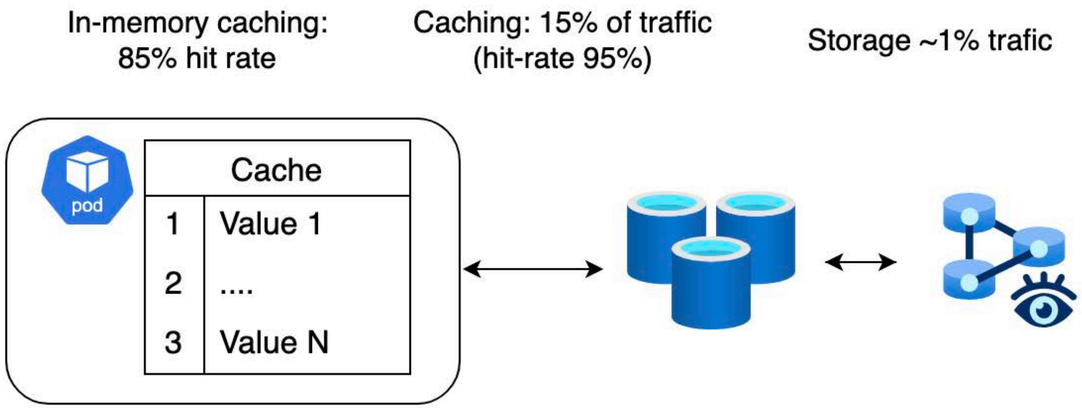
- For external storage choose
  - backward compatibility
  - size-efficient
- For pod (in-memory storage) choose
  - high-performance / low resources
  - size-efficient



Economy and Risk: Density vs. Stability

- Advantages
  - less trafic
  - more resource-efficient
  - less pods / nods you need
- Disadvantages
  - complex system





Life with compromises

- Team Cognitive Load
- Operational Complexity
- Invalidation and Consistency
- Risk of Cascading Collapse

Designing a Read-Intensive Architecture for Information Systems with High Data Throughput

Building high-load systems isn’t just adding resources. Efficient storage, caching, and network optimization are key, while well-designed data schemes and architectural decisions ensure performance and SLA compliance at scale

Ivan Sinitsin