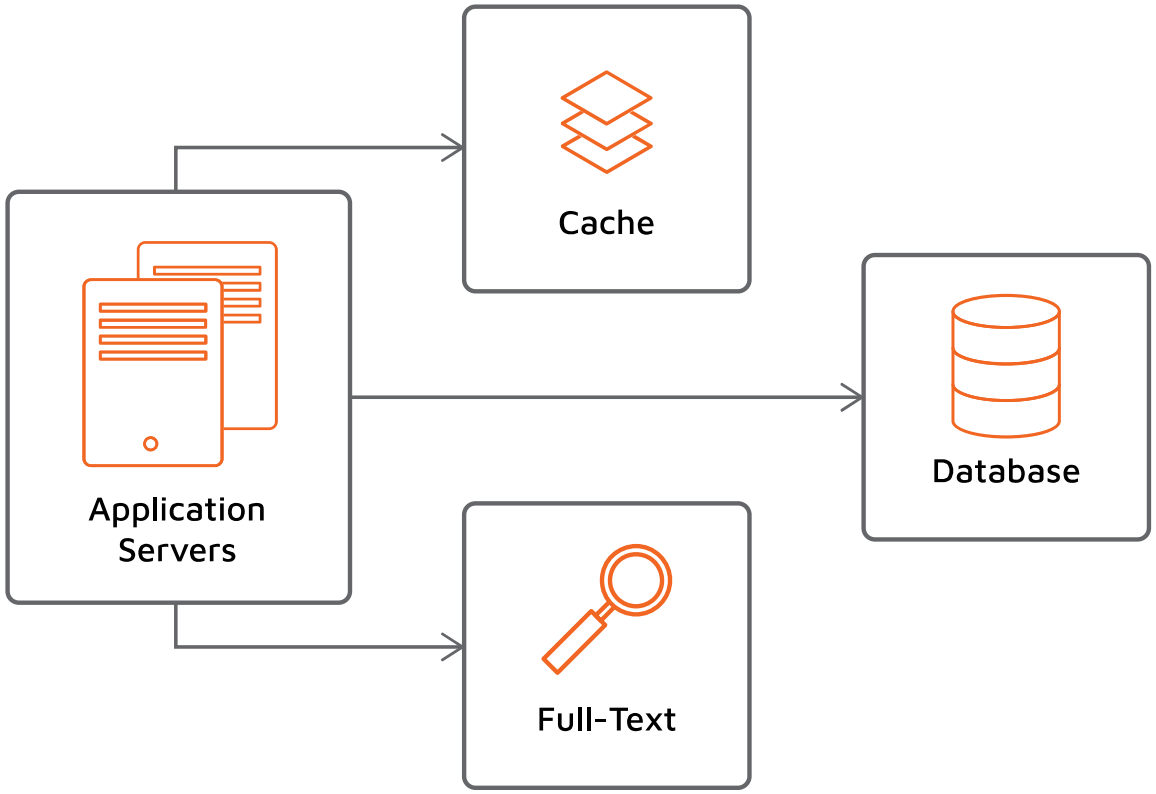
GridGain

# Боремся с сетевым оверхедом в распределённых системах: альтернативный подход к работе с данными

Pavel Tupitsyn
Lead .NET engineer at GridGain
.NET client maintainer at Apache Ignite
ptupitsyn@apache.org, ptupitsyn.github.io

# Traditional Approach: Bring Data To Code

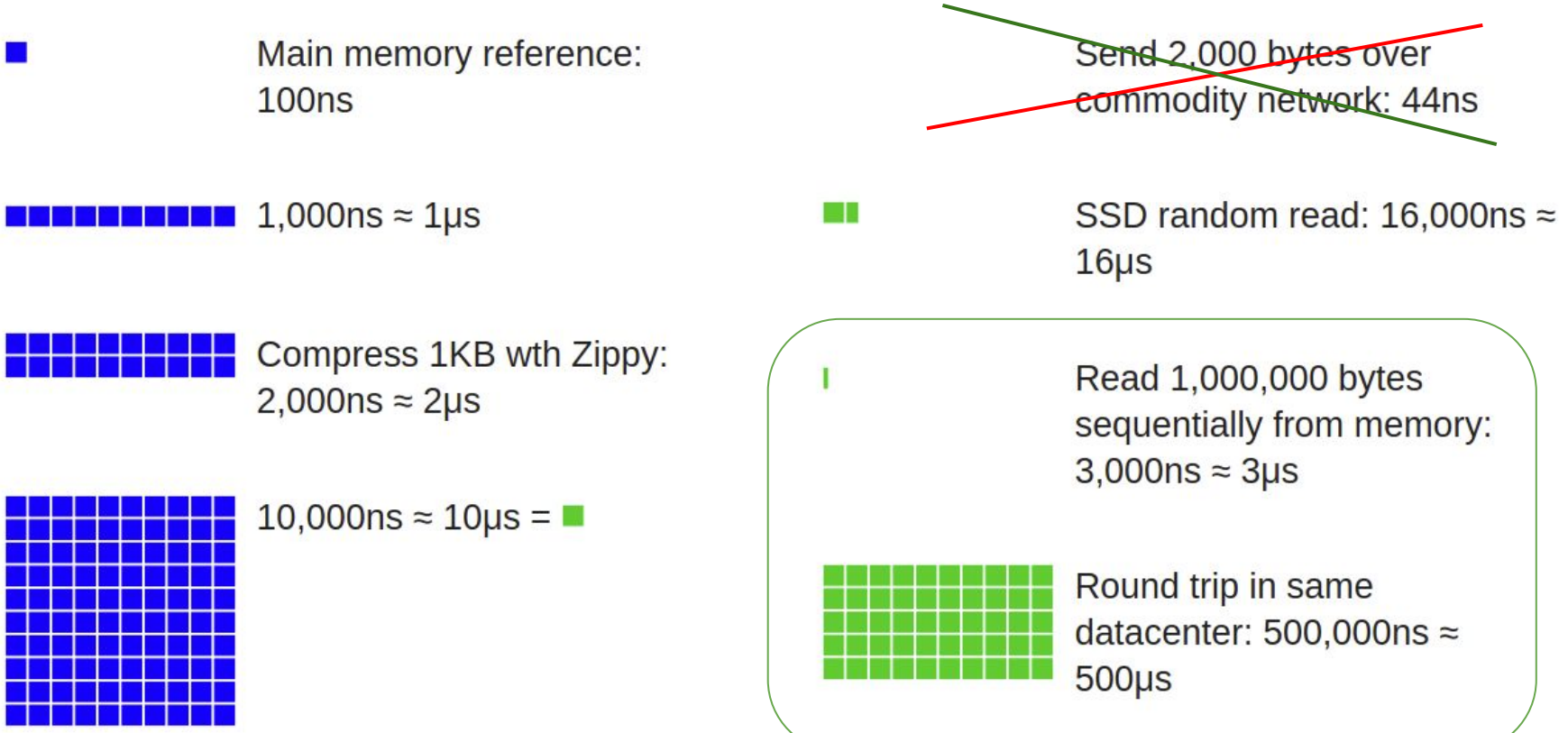



**A lot of network calls for one user search query:**

- Get item ids from ElasticSearch
- Get item descriptions from Redis
- If not in Redis, get from DB, update Redis

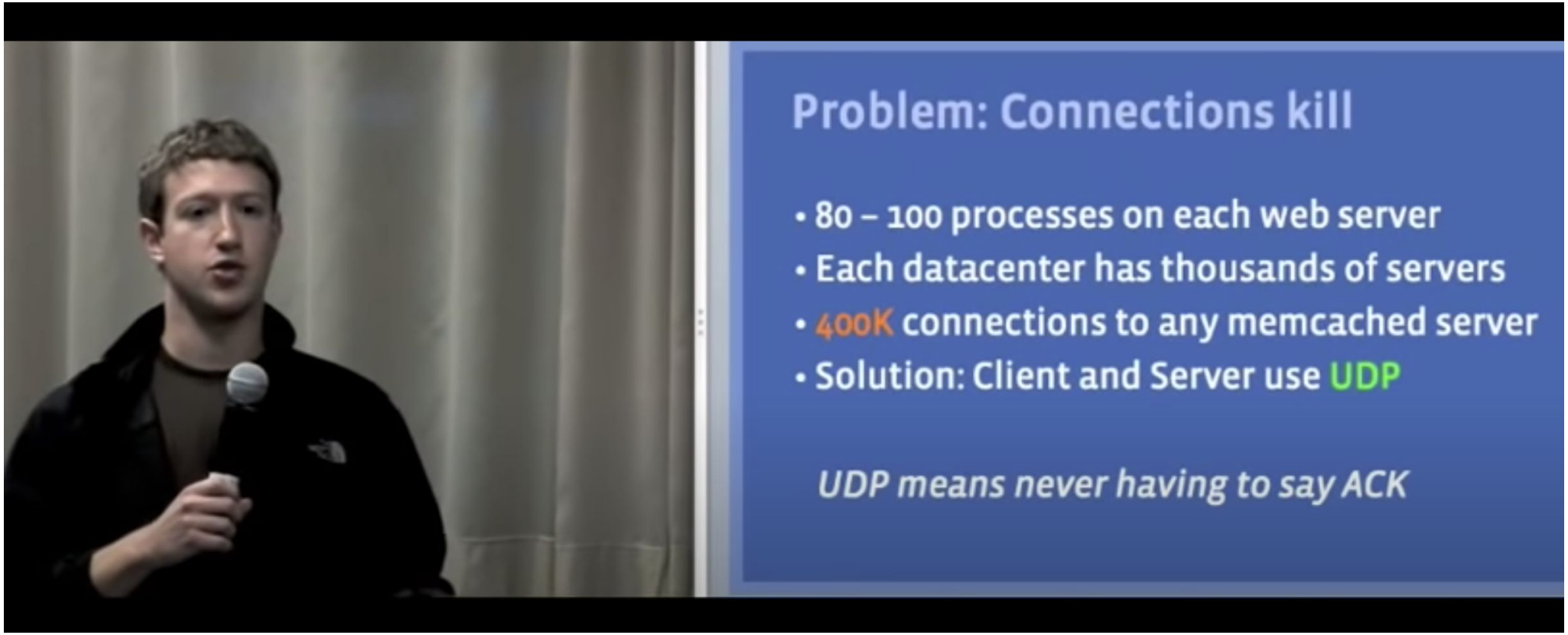
# But Network Is Fast! Or Is It?

Latency Numbers Every Programmer Should Know



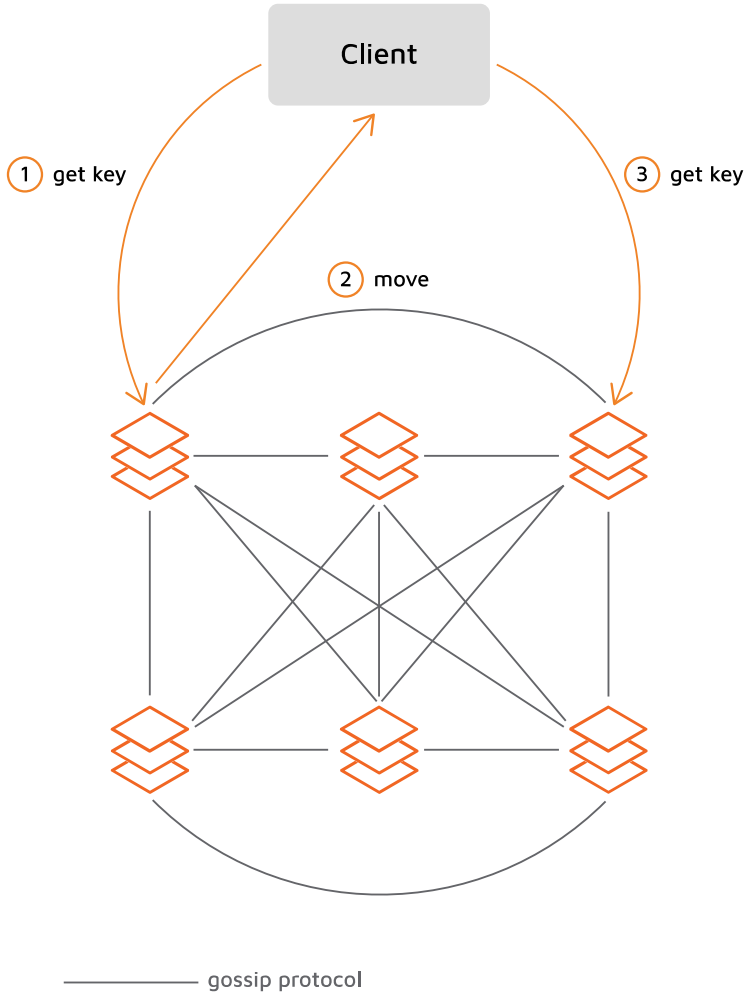

https://colin-scott.github.io/personal_website/research/interactive_latency.html

# Connections Are Not Free

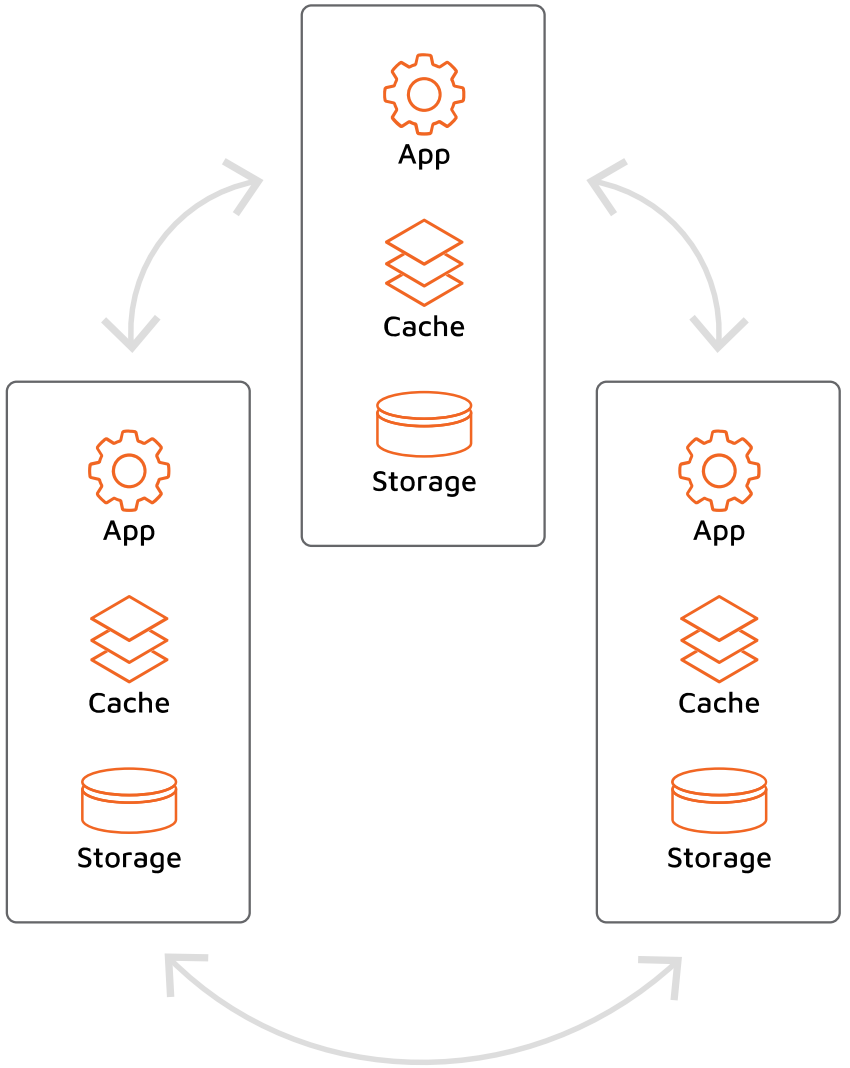



Facebook and memcached - Tech Talk
https://www.youtube.com/watch?v=UH7wkvcf0ys

# What if we store the data together with code?




VS




GridGain

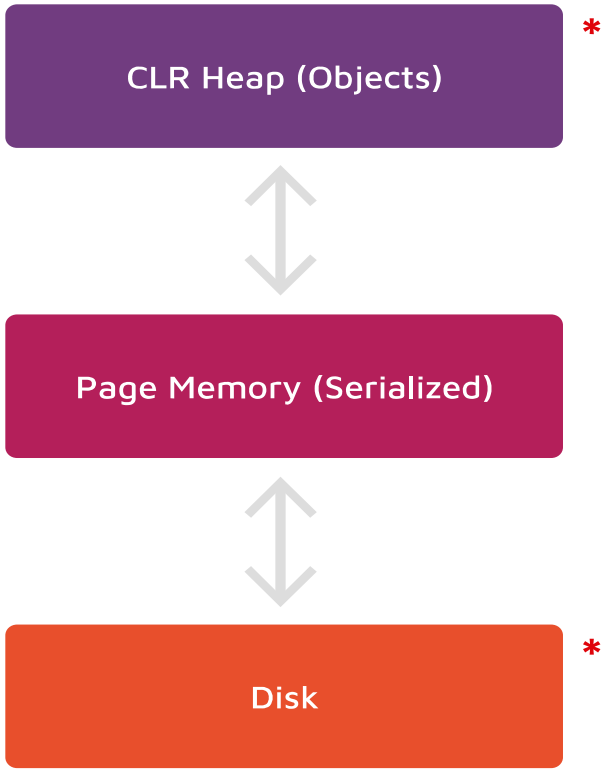
# Local Redis vs In-Process Ignite

| Method | Mean | Error | StdDev | Ratio | RatioSD |
|---|---|---|---|---|---|
| GetIgniteClrHeap | 46.33 ns | 0.305 ns | 0.286 ns | 1.00 | 0.00 |
| GetIgnite | 1,929.56 ns | 36.238 ns | 33.897 ns | 41.65 | 0.84 |
| GetRedis | 60,080.39 ns | 536.235 ns | 475.358 ns | 1,296.69 | 11.13 |

i7-9700K, Ubuntu 20.04, .NET 5, StackExchange Redis 2.2.4, Apache Ignite 2.9.1

# What is Apache Ignite?

- Distributed Database
- Transactional
- Automatic data partitioning (sharding)
- In-Memory + On Disk
- NoSQL + SQL / LINQ + FullText
- Streaming, messaging (pub/sub)
- Compute (map/reduce)
- Embedded Mode for .NET, Java, C++

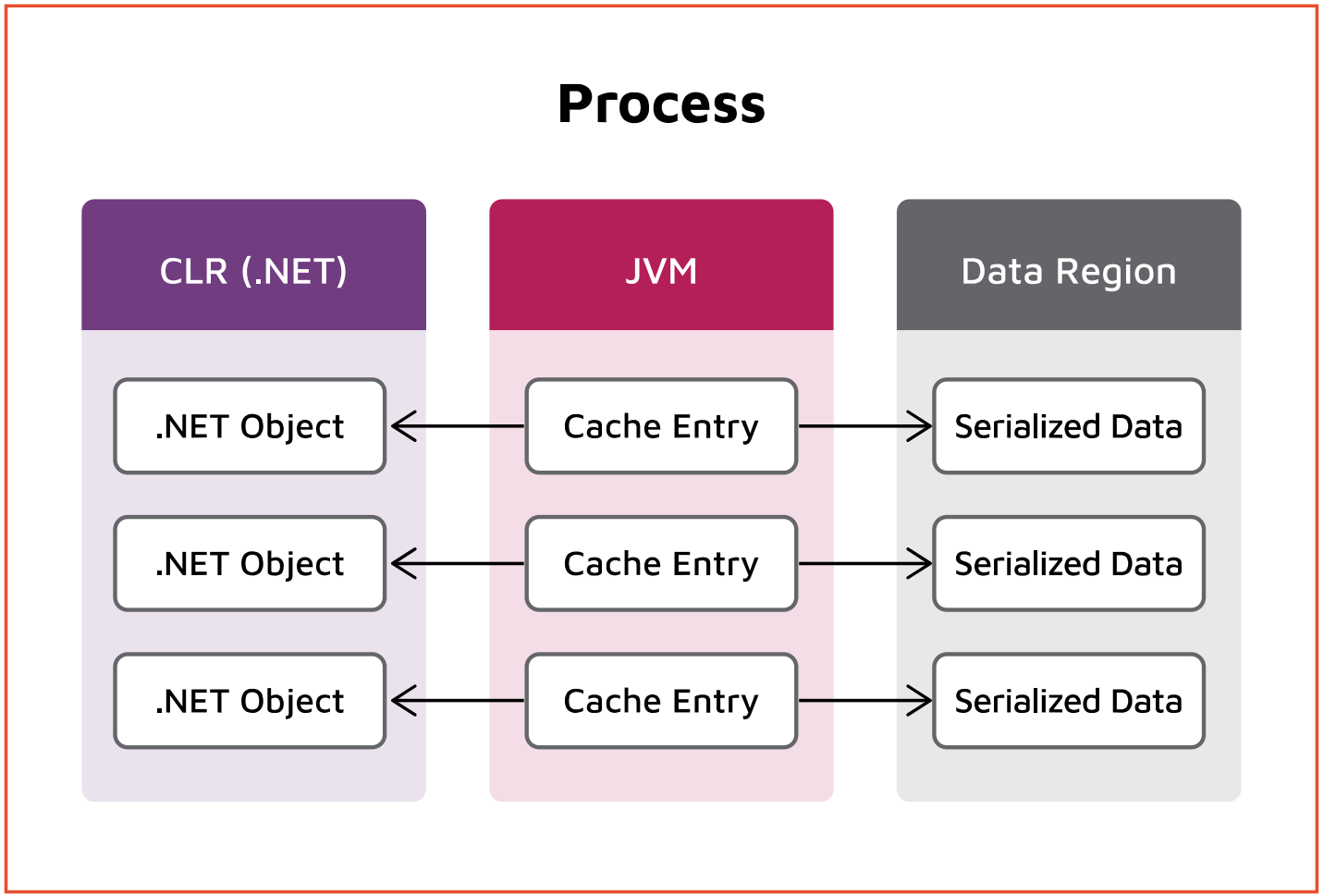


# Apache Ignite: Embedded Mode

- App + DB in one process (like SQLite)
- Data is in the process memory (“Page Memory” - unmanaged)
- Optionally on local disk
- Ignition.Start(config) to run
- Your app is the node: run on multiple machines to form a cluster
- Nodes find each other (address list / multicast / k8s discovery)
- Embedded mode is not the only choice:
  - Thin clients (.NET, C++, Java, JS, Python, PHP, Rust..)
  - ODBC
  - REST

https://ignite.apache.org/docs/latest/memory-architecture

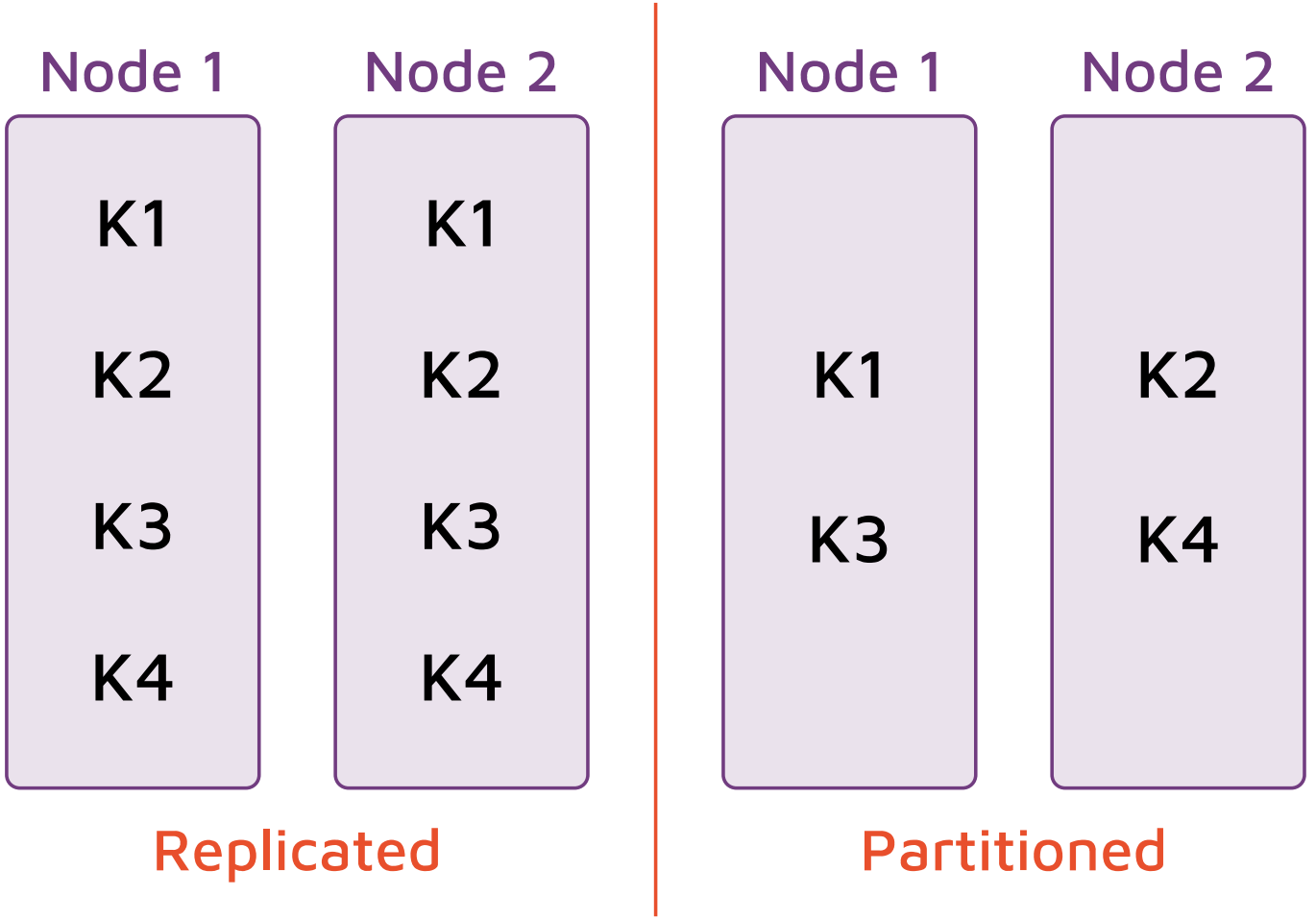
# Demo: Caching



# Caching Summary

- In-process cache avoids I/O costs
- Platform (CLR) cache reduces serialization costs
  - When writes < reads
- Replicated cache survives node loss

# Replicated and Partitioned Storage



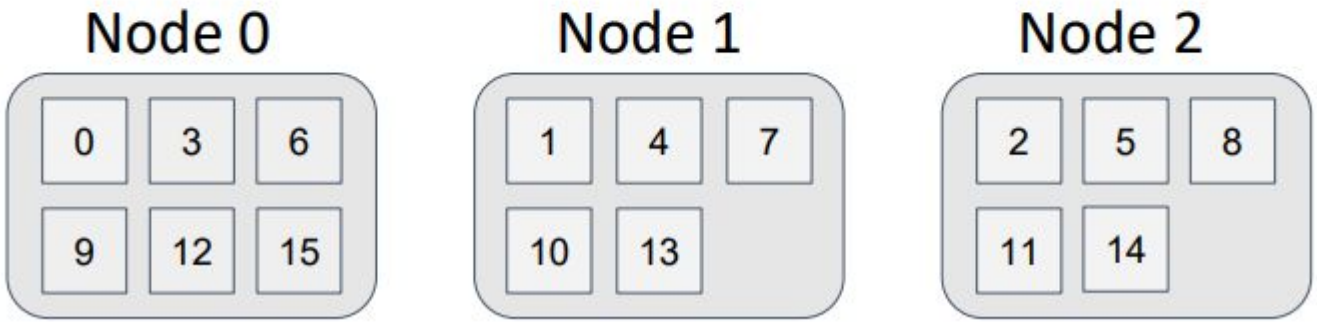

https://ignite.apache.org/docs/latest/data-modeling/data-partitioning
https://en.wikipedia.org/wiki/Rendezvous_hashing

# Partitioned Mode Explained

***h(K) mod N***

where

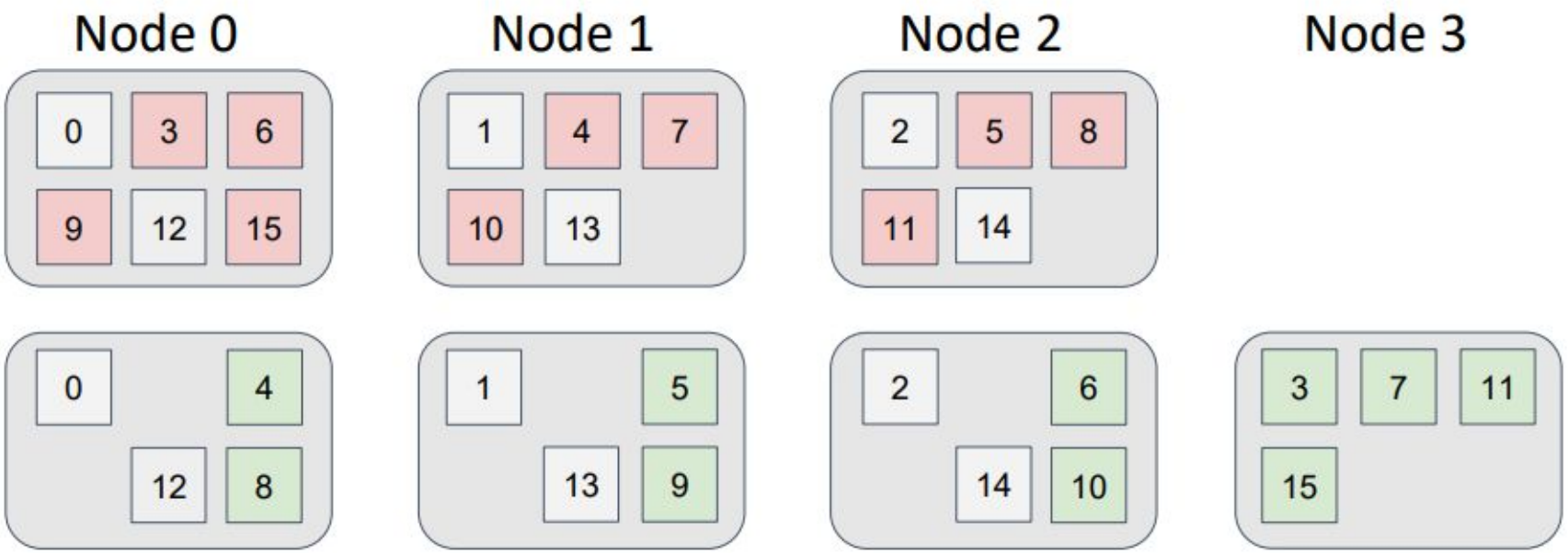
- *K* - object key
- *h* - hash function
- *N* - amount of nodes



# Partitioned Mode Explained

***h(K) mod N***

Add new node and ...



# Partitioned Mode Explained

**Distribution example: 4 keys, 3 nodes initially**

| | | | |
|---|---|---|---|
| K0 | Node 2 | Node 0 | Node 1 |
| K1 | Node 1 | Node 2 | Node 0 |
| K2 | Node 0 | Node 2 | Node 1 |
| K3 | Node 2 | Node 1 | Node 0 |

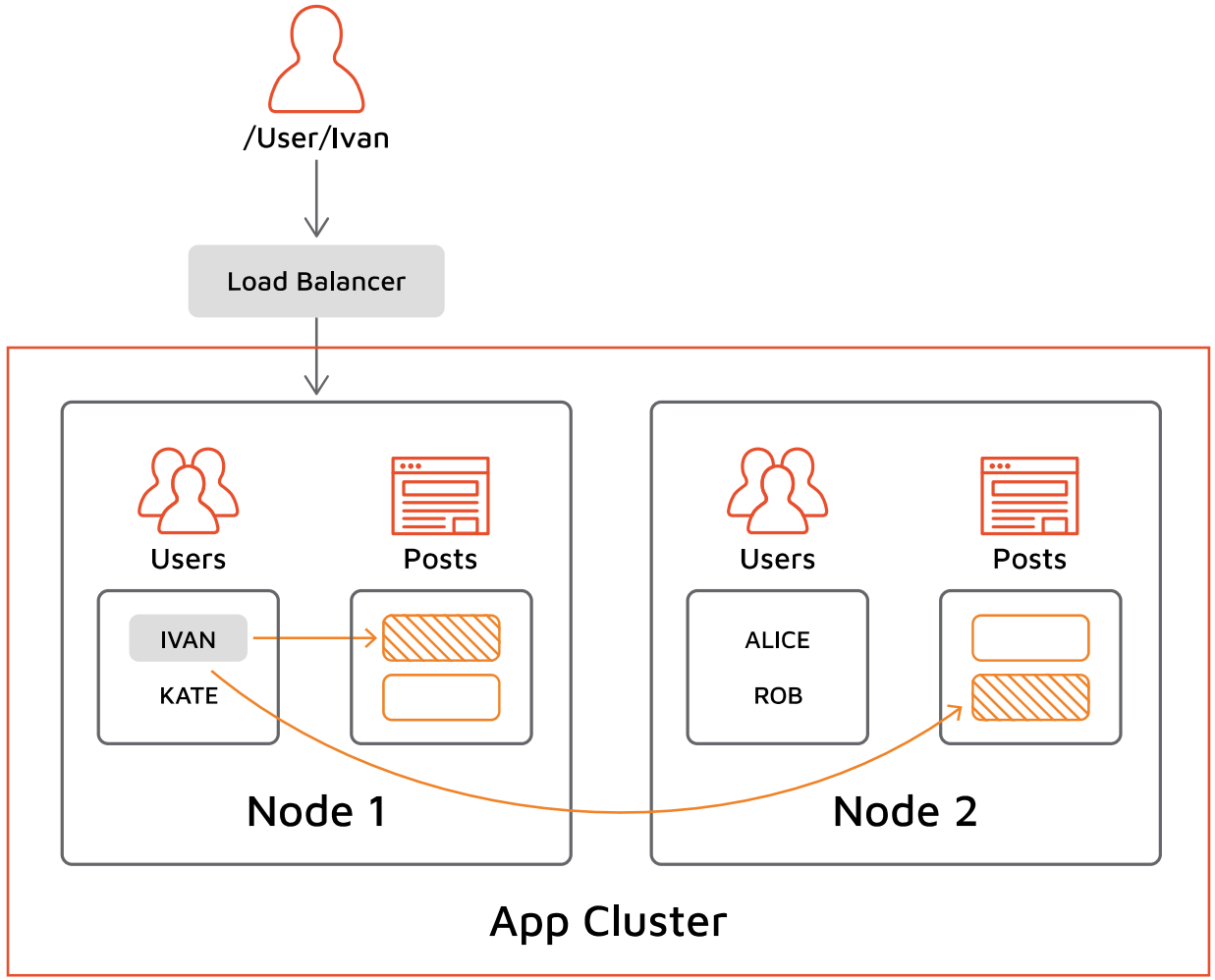
**Distribution example: 4 keys, new node added**

| | | | | |
|---|---|---|---|---|
| K0 | Node 2 | Node 0 | Node 1 | Node 3 |
| K1 | Node 1 | Node 2 | Node 3 | Node 0 |
| K2 | Node 0 | Node 3 | Node 2 | Node 1 |
| K3 | Node 3 | Node 2 | Node 1 | Node 0 |

```
nodes.OrderBy(n => HashCode.Combine(n, key))
```

https://en.wikipedia.org/wiki/Rendezvous_hashing

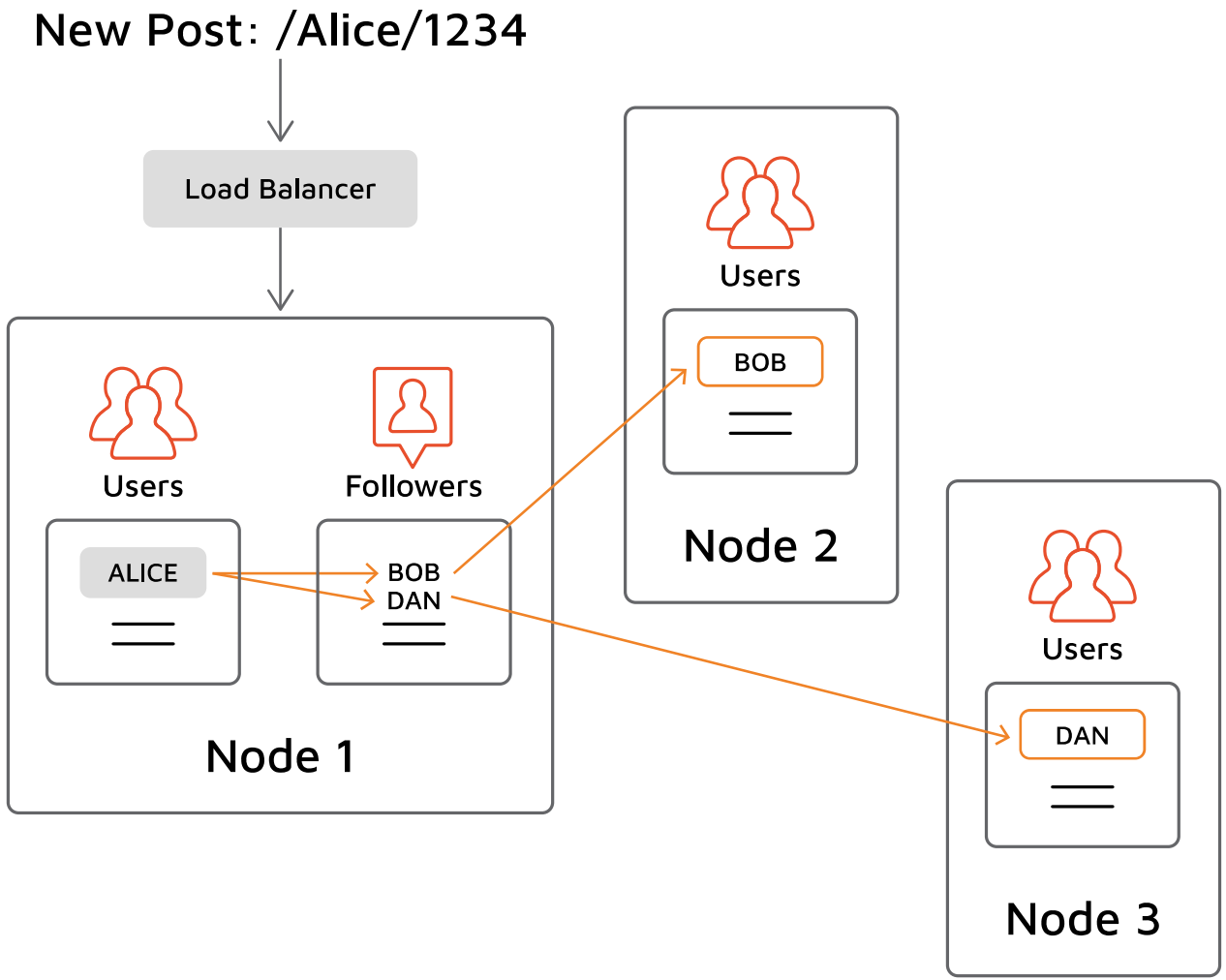
# Demo: Data Colocation



# Data Colocation Summary

- Partitioned (sharded) mode distributes data across nodes based on Key by default
- Use AffinityKey or [AffinityKeyMapped] to distribute data based on a different field
- Keep related data together
- You can’t colocate everything - optimize for important scenarios

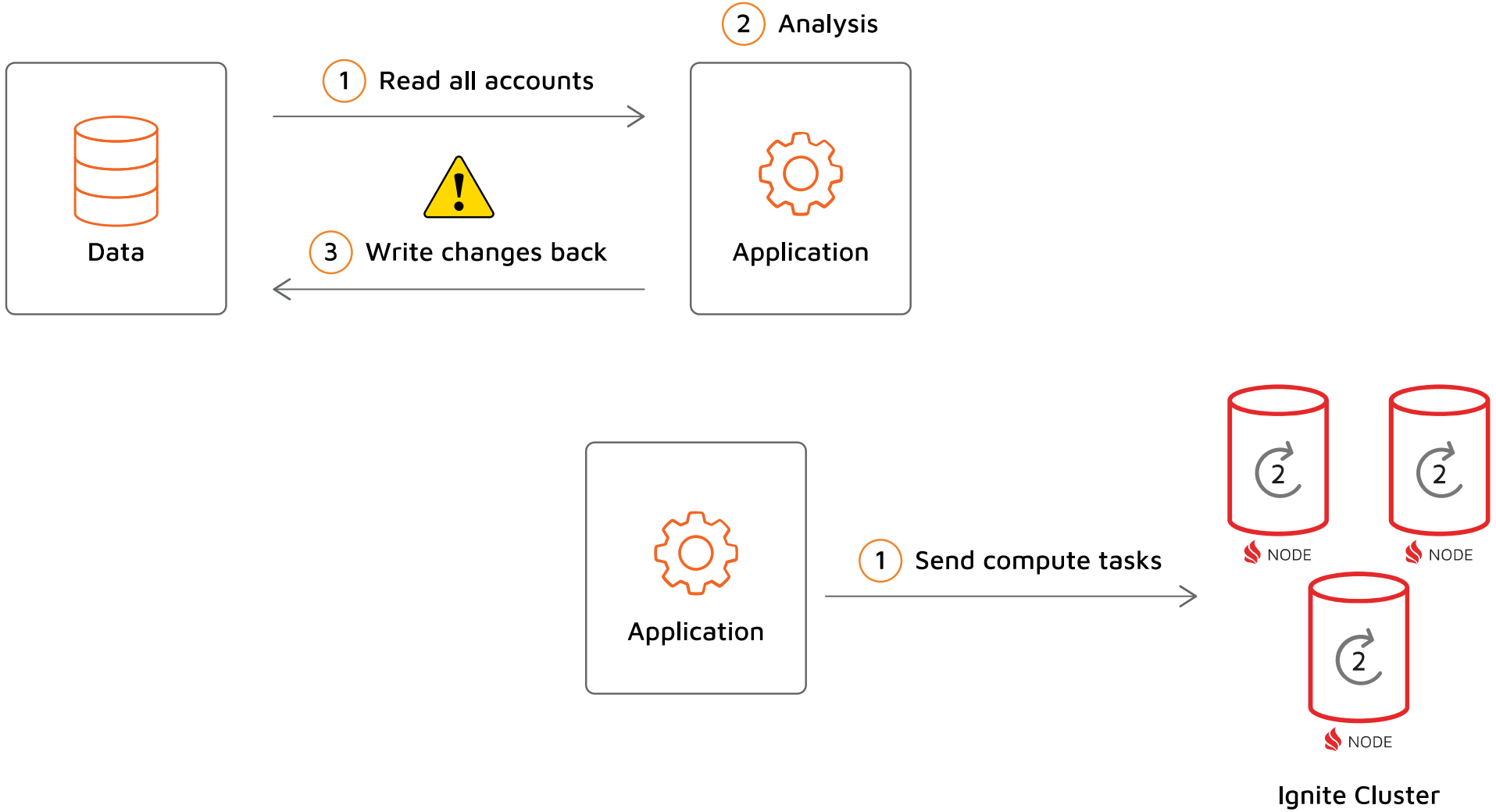
# Demo: Send Code To Data - Invoke



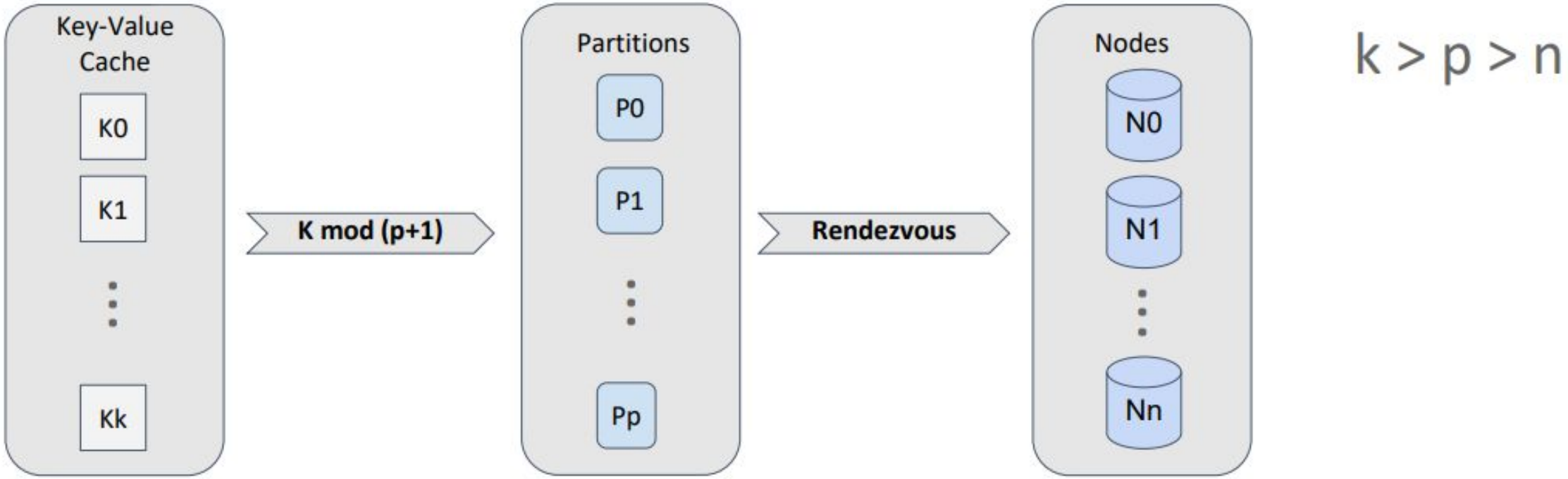
# Send Code To Data Summary

- Similar to SQL, e.g. “SELECT AVG ...”
- Not all computations can be expressed in SQL

# Send Code To Data: Compute



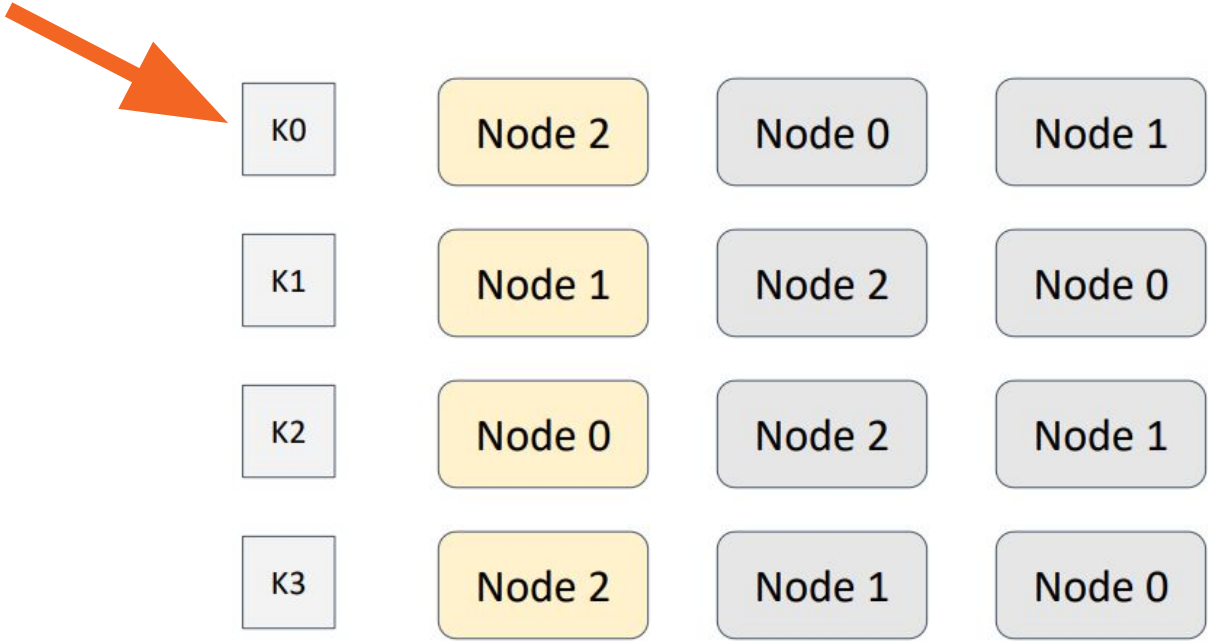
# Partitioned Cache: What is a Partition?




- Similar to hash table bucket

# Partitioned Mode Explained - Reminder

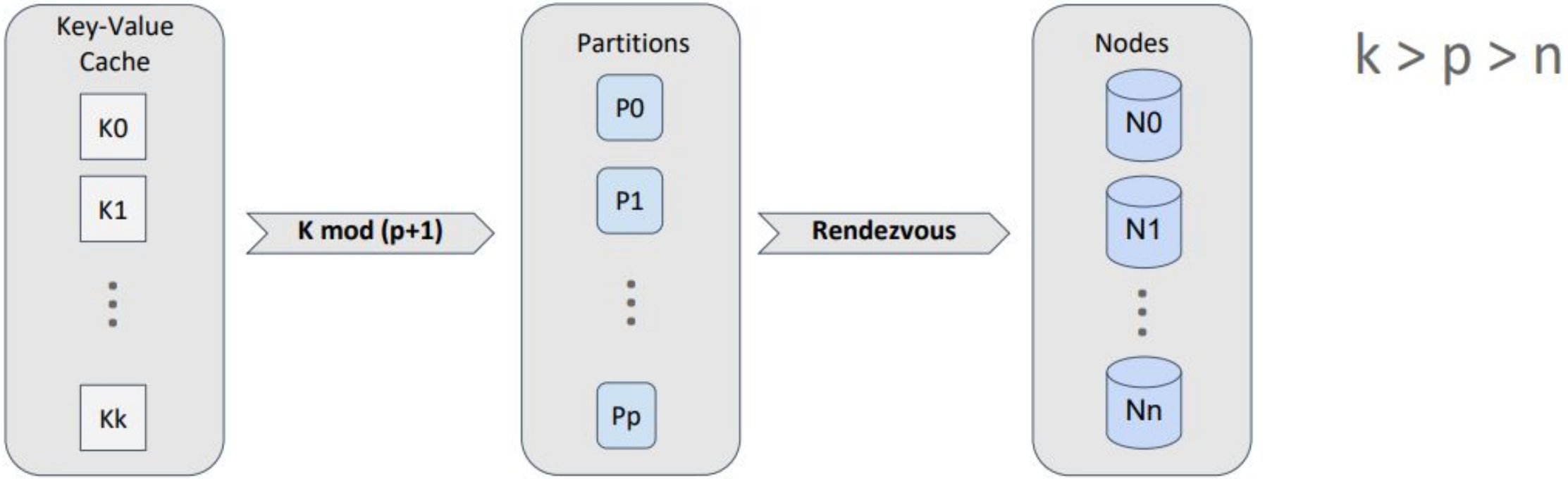
Partitions




```
nodes.OrderBy(n => HashCode.Combine(n, partition))
```

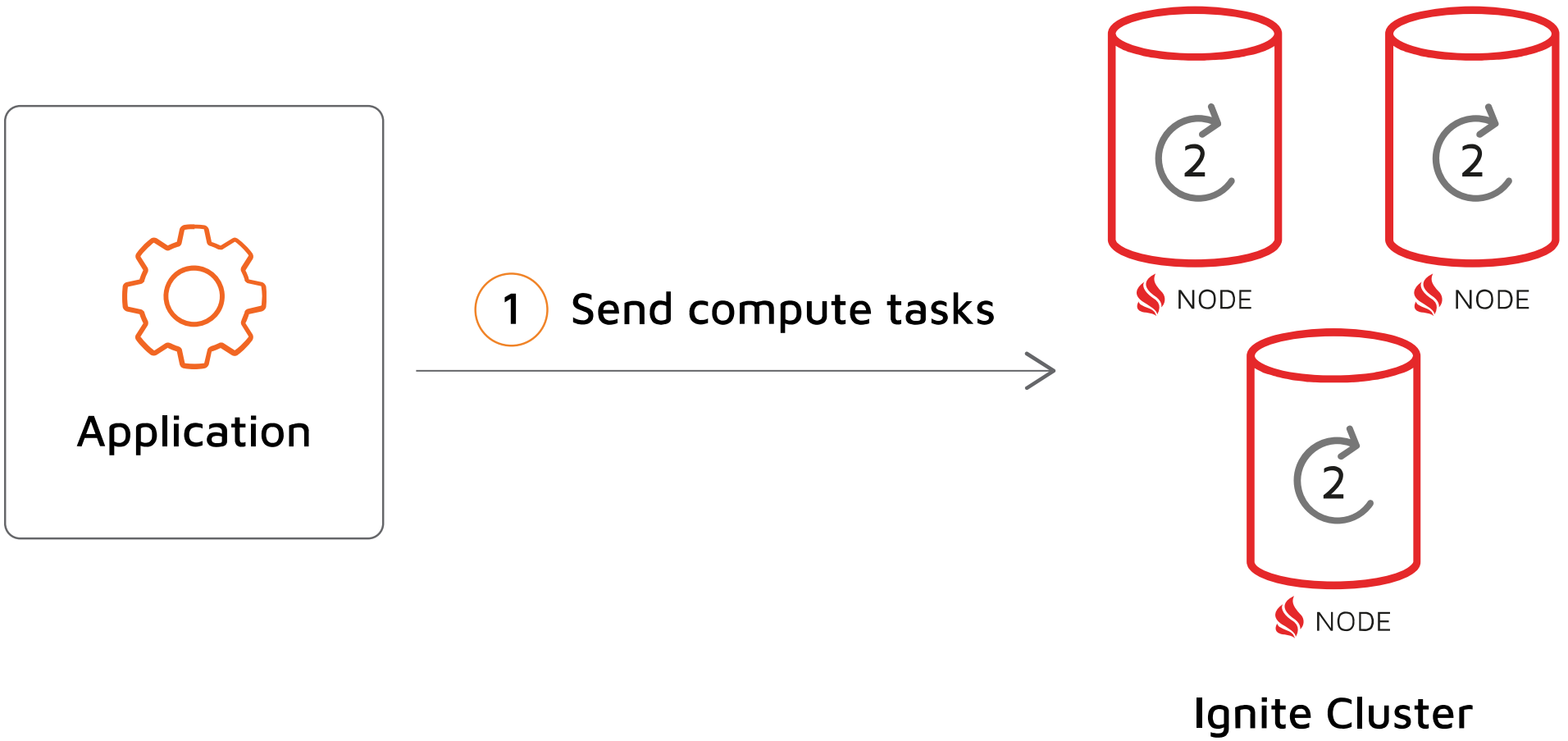
https://en.wikipedia.org/wiki/Rendezvous_hashing

# Partitioned Cache: What is a Partition?




- Simplifies per-key operations: **mod**
- Fixed number of partitions => distribution can be cached once per topology change
- Partition is moved as a batch
- Partition can be locked in place during the computation to guarantee consistency

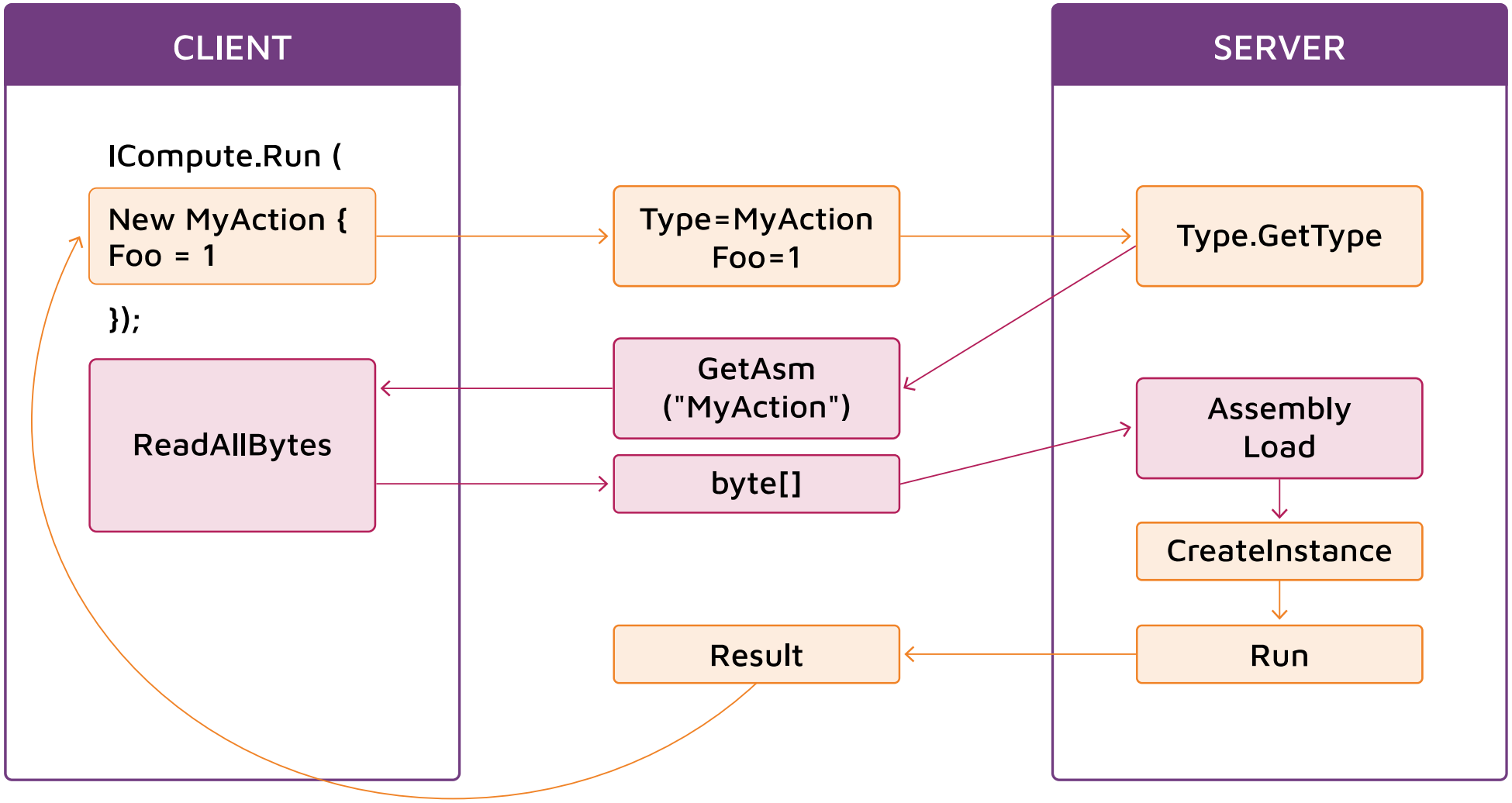
# Send Code To Data: Demo



# Send Code To Data: Summary

- Send code to when doing batch updates, changing large records
- Do not use for small changes - always take data size into account
- Topology may change and will change: use Affinity* APIs - they provide guarantees

# Send Code To Data: How It Works



# Summary

- Network calls are not free - far from it
- Usual approach with App and Data in separate clusters = overhead
- Combined (embedded) approach reduces this overhead
- Ignite simplifies combined approach
  - Automatic data distribution
  - Fault tolerance
  - Efficient APIs
- Choose data distribution strategy per use case
  - Replicated - more reads than writes, data fits in memory
  - Partitioned - data does not fit, more writes than reads
  - Backup copies
- Colocation: keep related data together
- Send code to data

# Q&A

- Распределение данных в Apache Ignite
  - https://habr.com/ru/company/gridgain/blog/489962/
- Rendezvous Hashing
  - https://en.wikipedia.org/wiki/Rendezvous_hashing
- Demos and Benchmarks
  - https://github.com/ptupitsyn/talks/
- Apache Ignite
  - https://ignite.apache.org

Pavel Tupitsyn, GridGain
ptupitsyn@apache.org
ptupitsyn.github.io